CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
COMPASS Reference Manual

REVISION RECORD

REVISION	DESCRIPTION
A	This manual obsoletes all previous editions.
(10-9-68)	
B	This manual updates COMPASS to the level of SCOPE 3.1.5.
(3-20-69)	
Publication No. 60190900	

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA 94304

or use Comment Sheet in the back of this manual

CONTENTS

INTRODUCTION 1

1.1 COMPASS DEFINED

6400/6500/6600 COMPASS, a comprehensive assembly system, provides a symbolic program language for the CONTROL DATA® 6400/6500/6600 computers. COMPASS is designed for efficient utilization of all computer resources and maximum flexibility in program construction.

COMPASS expresses symbolically all hardware functions of the 6400, 6500, and 6600 computers. The following features enable the programmer to control the assembly process:

- Free-field source format
- Assembly-time access to symbol table information
- Programmer control over local and common code blocks
- Macros (both programmer and system defined)
- OPDEF, a special macro form for redefining machine mnemonics
- Micro coding

1.2 COMPUTER HARDWARE CONFIGURATION

The basic computer includes one or two central processors, 10 peripheral processors (PP), 12 channels to which input/output devices can be connected, and central memory. The central processor (CP) is a high speed arithmetic device which handles the CP computational load programs held in central memory. The 6400 and 6500 systems have a unified arithmetic unit for sequential execution of instructions; the 6600 computer has ten arithmetic and logical units for simultaneous execution instructions.

Central memory (CM) stores executable programs together with the data the programs require.

Each of the ten peripheral processors (PPs) has separate memory and can execute programs independently of each other or the CP. The PPs transfer the programs to be executed by the CP from peripheral equipment to central memory and transfer input data as required. Similarly, PPs transfer output data generated by the central programs from central memory to peripheral equipment. The difference in functions associated with the CP and PPs coupled with a different instruction word-size capacity has resulted in the development of two distinct operation code sets.

1.3 OPERATING SYSTEM

COMPASS operates under control of the SCOPE operating system, which is in constant control of all jobs, handling storage allocation, job scheduling, accounting, I/O control and operator communication.

PROGRAM STRUCTURE AND ORGANIZATION 2

2.1 CP AND PP CODING

A COMPASS program consists of either central processor (CP) code or peripheral processor (PP) code. The machine instructions for the two processors are different and may not be intermixed within a program, but most of the pseudo instructions are used in both CP and PP programs. Pseudo instructions may differ in specification, significance, or result, according to whether the program is a CP or PP program.

2.2 SUBPROGRAM STRUCTURE

The programmer or COMPASS assigns to each subprogram one or more local or common blocks into which all code is assembled. A local block contains code accessible to the subprogram only; a common block contains code accessible to all subprograms loaded together. A program may use a maximum of 252 local and common blocks in addition to those defined by the assembler.

As assembly proceeds, all locations and references to locations within a block are considered relative to the start of that block. COMPASS maintains the origin of each block, the current position within each block, and the final length of each block. The programmer may manipulate origin, location, and position counters to control position. At the end of assembly, COMPASS assigns an origin, relative to the start of the first program block, to each local block in the order in which its name was introduced. The length of a subprogram is the sum of the maximum origin counter values of all local blocks.

2.2.1 LOCAL BLOCKS

Code within local blocks is accessible only to the subprogram itself. Three local blocks, pre-defined by COMPASS in every subprogram, need not be declared by the programmer:

Absolute block, used for all absolute code

Zero block, used by COMPASS when no programmer assigned block is specified in a relocatable CP assembly

Literals block, contains all literal data values

The absolute block is the nominal block for all absolute subprograms as well as the block for absolute origins in relocatable subprograms. The zero block is the nominal block for all relocatable subprograms. PP subprograms are always absolute; CP subprograms may be absolute or relocatable. All code in a subprogram will be in either the zero block or the absolute block, unless the programmer requests or uses another block. The programmer may refer to the zero block in a USE statement, he may refer to the absolute block only with the ORG statement.

All data literals are assigned to the literals block which may not be referenced by the programmer. At the end of assembly, the literals block is assigned an origin at the end of the zero block.

The programmer may define and use additional local blocks with the USE statement. Named local blocks are considered extensions of the zero block; they are assigned origins by COMPASS at the end of the zero block (after any literals), in the order in which they are declared.

2.2.2 COMMON BLOCKS

Code assigned to common blocks is accessible by all subprograms loaded together. Common blocks are assigned origins by the loader at load time (unlike local blocks which are assigned origins by COMPASS at assembly time). They may bc labeled common or blank common. Labeled common includes blocks designated with a numeric name.

Data may be pre-loaded into labeled common but not into blank common. Space may be reserved in blank common by using only the BSS or ORG pseudo instructions.

2.3 COUNTERS

Origin, location, and position counters are maintained by COMPASS to define the location of code and the current position within a word. These counters may be reset by pseudo instructions, and their values may be tested at any point.

2.3.1 ORIGIN COUNTER

COMPASS maintains the origin counter to indicate the location of loader-placed instructions. For each block, the origin counter starts at zero relative to the block origin or at the last known size of that block if it has been previously used.

The origin counter is incremented by one for each completed word of assembled data. Its value may be reset with the ORG pseudo instruction. When the special element *O is selected, the current value of the origin counter is the value used.

2.3.2 LOCATION COUNTER

Normally, the location counter has a value identical to the origin counter and gives definition to location symbols. It may be adjusted, however, to differ from the origin counter if succeeding data is to be executed in a memory area different from its assigned load time area. For example, a block loaded in ECS might be subsequently moved and executed in another area. The location counter should reflect the actual location at which execution occurs.

The location counter may be reset with the LOC pseudo instruction. When either of the special elements * or *L is selected, the current location counter value is the value used.

2.3.3 POSITION COUNTER

This counter maintains a position within a 60-bit or 12-bit word of assembly. As each code generating instruction is encountered, the position counter is updated to reflect the next available bit position. The position counter contains the number of the high order bit of the field, numbered from 59 to 0. In CP instructions, it has a value of 59, 44, 29, or 14. In PP instructions, 11 is the normal value. These values may be modified by the VFD pseudo instruction (section 5.6.4).

Whenever the special element $ is selected, the current position counter value is used.

2.4 FORCING UPPER

In central processor assemblies, assembled data is packed sequentially into a 60-bit word in bytes of 15, 30, or 60 bits. If there is not room in a partially filled 60-bit word for the instruction or data currently being evaluated, the remainder of that word is filled with 15-bit no-operation instructions (46000_8), and the current instruction is assigned the first position in the next word. Packed data can be manipulated with the VFD pseudo instruction.

COMPASS also forces upper when any of the following occurs:

- A symbol or + appears in the location field of the current statement
- Current instruction is PS, RE, WE, or XJ, unless the location field contains a minus sign
- Current instruction is one of the pseudo instructions END, LOC, BSS, BSSZ, DATA or DIS. ORG also forces upper in the block which it references (section 5.2.2).

Forcing upper is automatic after JP, RJ, PS, XJ and an EQ or ZR with a single address (the unconditional EQ or ZR). The ECS instructions WE and RE must appear in the upper 30 bits of an instruction, and, when executed successfully, execution continues at the beginning of the next 60-bit word. The lower half of the WE or RE word presumably contains a jump to an error routine to be taken if WE or RE is rejected. COMPASS does not force upper after WE or RE.

In a PP assembly, no forcing upper occurs; a + in the location field is ignored except on a VFD line, the position counter is reset to the beginning of a PP word.

Automatic forcing upper after JP, RJ, PS, EQ, and ZR as well as forcing upper on PS, RE, WE, or XJ can be negated by using a minus sign in the location field of the next instruction. When a minus sign appears, the current line is assembled into the next position large enough to contain it.

COMPASS LANGUAGE CODING 3

3.1 SOURCE STATEMENTS

3.1.1 CODING FORMAT

A COMPASS program consists of a sequence of symbolic statements. Each statement contains a maximum of four fields in the order listed below. The format is essentially free field.

Location field must begin in column 1 or 2.

Operation field may begin in any column from 3 to 35.

Variable field must begin before column 36.

Comments field may begin after the termination of the variable field, or no earlier than column 36 if the variable field is empty.

Columns 73-90 may be used only for comments; generally they are used for sequencing. Columns 81-90 are used for sequencing by library maintenance programs; they are normally not used by the programmer.

Fields are separated by one or more blanks. Blanks are interpreted as field separators except when embedded in the comments field, in character data items, or in a parenthesized macro parameter.

A statement may be a comment or an instruction; it may contain as many as ten 90-column lines. Column 1 indicates the type of line: an asterisk identifies a comment statement; a comma indicates a continuation of the previous line. Any other character in column 1, including blank, indicates the beginning of a new statement.

A comment statement may be introduced either by an asterisk in column 1 or by blanks in columns 1-35. Comment lines are listed in assembler output; they have no other effect on assembly.

A line introduced by a column 1 comma is considered a continuation of the preceding line. A maximum of 9 continuation lines are permitted. Column 2 of each continuation line is interpreted as an immediate continuation of column 72 of the preceding line. The break between lines need not coincide with a field or subfield separator; even a symbol may be split between the two lines. Continuation lines beyond the ninth are considered comments.

A line with an entry in the location field but not the operation field creates a word of zeros and is equivalent to the instruction:

```
loc     BSSZ    1
```

Example of a standard format for source lines:

Column	
1	Blank, asterisk, or comma
2-9	Location field, left justified
10	Blank
11-16	Operation field, left justified
17	Blank
18-	Variable field, terminated by 1 or more blanks
36-	Comments field

3.1.2 CATENATION AND MICRO SUBSTITUTION

Any line not containing a column 1 asterisk is examined for the two special characters → and ≠ before COMPASS attempts any other interpretation. The catenation character → indicates that two adjoining columns are to be linked (section 6.1.2). The ≠ mark indicates micro substitution (section 7). The line which is changed as a result of catenation or micro substitution may be any type: a comment line, an instruction, or a continuation of an instruction.

Substitution may require generation of continuation lines or cards. The free field format generates continuation cards automatically. During catenation or micro substitution COMPASS preserves as many blanks as were written between fields and subfields; the original columnar arrangement of fields can be altered after substitution occurs. Micro substitution might itself cause a continuation line to be produced.

Statements which are part of definitions (section 3.1.3) are not examined for the two special marks ≠ and →. For this type of statement, catenation or micro substitution occurs at the time of execution rather than at the time of definition. Therefore, an ENDM cannot be created which would terminate a micro definition by using micro substitution or catenation.

Catenation and micro substitution do occur on lines which are being skipped.

3.1.3 STATEMENT TYPES

Statements processed by COMPASS fall into three categories:

- A normal statement which is assembled and may produce output
- A statement which is bypassed because of a conditional instruction test which failed
- A statement which is part of a definition: those lines contained between a MACRO and ENDM, between a DUP and ENDD, between a RMT and a terminating RMT

3.2 INSTRUCTION ELEMENTS

Location Field

The location field may be blank or may contain one of the following:

Symbol

Name

+

-

Operation Field

The operation field must be present, and may contain one of the following:

Central processor operation code

Peripheral processor operation code

Pseudo instruction

Macro name

Variable Field

Contents of the variable field are dictated by the operation code. For COMPASS machine instructions, this field consists of one, two, or three subfields separated by commas. A subfield in CP instructions may contain register names separated by the operators + - * / . COMPASS determines the octal value of the instruction from these operators; they may not be replaced by any other characters.

Comments Field

This field is optional and may contain any combination of characters. The catenation mark (→) and the micro mark (≠) produce the same results in the comments field as in any other field.

3.3 SYMBOLS

A symbol is a sequence of 1 to 8 characters representing a value. Symbol value is determined according to its use as follows:

> In the location field of a machine instruction and certain pseudo instructions, the value assigned to the symbol is the current value of the location counter.
>
> In the location field of an EQU or SET pseudo instruction, the value in the address field is assigned to the symbol.
>
> In a list of external symbols, both symbol definition and value assignment are accomplished outside the bounds of the current program.
>
> By default. If the symbol is preceded by =S or =X and has no other definition, COMPASS defines it.

Absolute symbols may be defined with the EQU or SET pseudo instructions or as location symbols in code with an absolute origin. They are assigned a 21-bit value.

Relocatable symbols are assigned a value relative to an unknown base address either in common storage or within the subprogram. For the purpose of symbol definition, relocatable symbols may be represented in absolute code in all blocks other than the zero block.

Symbols acceptable to COMPASS may contain characters which are illegal identifiers under other systems such as UPDATE, COPYN. A symbol may not include any of the following characters:

> * / , + - or blank

The first character may not be $ or = or numeric. Other special characters must be used with care. In CP programs, a decimal point will produce a register name if the decimal point is the second character and A, B, or X is the first. Refer to macro definition rules in section 6.

A symbol in a CP assembly may not be An, Bn, or Xn, where n is a single digit from 0 to 7.

Examples of legal symbols:

A	A10	A1.75
A=B	AAAAAAAA	A(B)
ABCDEF.3	A$$$	.01

Some symbol names are further restricted if they are used as the following:

Subprograms names

External symbols

Entry points

Common block names

These are called linkage symbols since they are used by the loader. Such symbols must begin with a letter (A-Z), and may not exceed seven characters. PP subprogram names may begin with a letter or a number and may not exceed three characters.

3.4 REGISTERS

Register names are symbolic representations of the 24 operating registers. Register names are predefined in central processor COMPASS assemblies and may not be redefined in the program. They are of two forms:

An, Bn or Xn, n is a single digit from 0 to 7. Any other term for n is interpreted as a symbol rather than a register name.

A.n, B.n, or X.n, n may be a single symbol or an integer. If the value of n exceeds 7, it is truncated to the low order 3 bits and a warning flag is issued.

Register names of either form are considered ordinary symbols in a PP assembly.

Examples:

A1	Accumulator register 1
A10	Symbol, not a register name
A.1	Accumulator register 1
A.10	Accumulator 2; produces a warning flag ($10_{10} = 12_8$ which truncates to 2)

The following produce equivalent results:

```
SB3     A2+ALPHA                      SUM     SET     3
                                      SUB     SET     2
                                              SB.SUM A.SUB+ALPHA
```

3.5 DEFERRED SYMBOL DEFINITION

Definition of a symbol may be deferred until end of assembly. At that time, COMPASS defines all deferred symbols not defined by conventional methods.

Deferred symbols may be indicated in an address expression by the forms:

=Ssymbol normal relocatable symbol which results in the following:

> If a symbol is not defined, it represents a location which COMPASS assigns at the end of the zero block. All subsequent references to that symbol, whether preceded by =S or not, are to that assigned location. Any symbol so defined may not be used where a previously defined symbol is required.
>
> If the symbol is defined, COMPASS does not define it again as a deferred symbol. The programmer-defined value of the symbol is used instead.

=Xsymbol external symbol which results in the following:

> If the symbol is not defined, the symbol is assumed to be external as though declared in an EXT pseudo instruction. It must conform to the rules for linkage symbols.
>
> If the symbol is defined, it represents the value assigned by the programmer COMPASS does not define it again as a deferred external symbol.

If a symbol appears as both =S and =X, or as =X in an absolute assembly and has no other definition, it is undefined and produces an error.

3.6 NAMES

A name is a symbol which indicates one of the following:

block	instruction bracket
macro	micro

Names do not conflict with ordinary symbols since they are used differently. Names may not be used in address expressions but the rules for forming them are less strict than for ordinary symbols. A name may be any combination of 1 to 8 characters except blank or comma.

Examples of legal names:

2	3A	A+B*C
X*Y/Z	$+A	=48
2+6	*LA$+SF	1.5

3.7 ABSOLUTE DATA

Absolute data is used in literals, LIT and DATA pseudo instructions, and as constants in address expressions. COMPASS supplies a format for data specification which is common to all these usages, with minor exceptions.

Data item describes an absolute item which produces one or more full-word values. The following are data items:

A subfield of the DATA pseudo instruction

A subfield of the LIT pseudo instruction

A literal (the portion which follows = if the item
is not =Ssymbol or =Xsymbol)

Address constant describes a constant with a maximum length of 60 bits which may appear in an address expression. Constants appear in machine and pseudo instruction subfields, including VFD.

Absolute data may be character data or numeric data. Numeric data may be octal, decimal, single-precision floating point, double-precision floating point or fixed point.

3.7.1 CHARACTER DATA

Each character data item whether it is used as a data item or an address constant takes the following form:

$$n \begin{bmatrix} \mathrm{C} \\ \mathrm{H} \\ \mathrm{A} \\ \mathrm{R} \\ \mathrm{L} \end{bmatrix} \left\langle \text{character string} \right\rangle$$

n is a character count. The character string is justified within the given field length as follows:

C	Left justified with zero fill; 12 zero bits are guaranteed at end of string even if another word must be allocated
H	Left justified with trailing blanks
A	Right justified with leading blanks
R	Right justified with leading zeros
L	Left justified with trailing zeros

Field length for a character string is determined according to the following rules:

In data items (DATA, LIT, literals), the characters are justified within a 60-bit (CP) or 12-bit (PP) word.

In any EQU or SET address field, characters are justified within an 18-bit field.

In a VFD pseudo instruction, characters are justified within the field size specified by the VFD subfield.

As a constant in an address expression, characters are justified in a field which is equal in length to the address size (18 or 6 bits in CP; 18, 12, or 6 bits in PP).

In address expressions, the C and L options are handled the same; the 12 trailing zero bits are not guaranteed on C character strings in address fields.

The catenation character → is not converted to its octal equivalent (65) in a character data string.

The following characters are special and should not be included in character strings:

;	cannot be used; this character is used by COMPASS as an internal delimiter; in macro definitions as a parameter marker. A non-fatal error is issued when ; is used
→	produces catenation; symbol is eliminated
≠	produces a micro substitution if a legal character micro name is enclosed between two of these charaeters

COMPASS interprets the character string in a character data subfield according to the value of n.

(a) If n is missing, the programmer may specify delimiters for the character string:

$$\begin{bmatrix} \mathrm{C}\\ \mathrm{H}\\ \mathrm{A}\\ \mathrm{R}\\ \mathrm{L}\end{bmatrix} d \left\langle \begin{array}{l}\text{any character}\\ \text{string not}\\ \text{including d}\\ ; \rightarrow \text{ or } \neq\end{array}\right\rangle d$$

d is any single character. All characters between the first and second occurrence of d are considered the character string.

This form of character specification is restricted to data items (a DATA or LIT subfield or a literal) since address items beginning with an alphabetic character are considered symbols rather than constants.

A minus sign may precede C, H, A, R, or L to complement the character string.

(b) If n is zero, the character string is considered ended when a subfield terminator is encountered:

$$0 \begin{bmatrix} \mathrm{C}\\ \mathrm{H}\\ \mathrm{A}\\ \mathrm{R}\\ \mathrm{L}\end{bmatrix} \left\langle \begin{array}{l}\text{any character string not}\\ \text{including blank , } ; \rightarrow\\ \text{for data items, or } + - \text{ , } * \; / \; ; \rightarrow\\ \text{or blank for address constants}\end{array}\right\rangle$$

A blank or comma terminates this character string if it is used in LIT, DATA, or a literal (a data item). Blank, comma, + - * or / terminates an address constant in this format. When n is preceded by a minus sign, the character string is complemented.

When used as an address constant, the string may not exceed ten characters.

(c) If non-zero, n is the count of characters in the string.

$$n \begin{bmatrix} \mathrm{C}\\ \mathrm{H}\\ \mathrm{A}\\ \mathrm{R}\\ \mathrm{L}\end{bmatrix} \left\langle \begin{array}{l}\text{any characters}\\ \text{not including}\\ ; \rightarrow\end{array}\right\rangle$$

For address constants, $(1 \le n \le \frac{\text{field length}}{6})$. For data items, n may be any value. When n is preceded by a minus sign, the character string is complemented. For the C designation, the zero bits to be added are not included in the character count n.

If the character count for a data item is greater than the number of columns remaining, including maximum allowable continuation cards, an A error will result.

(d) Empty character strings:

In either case (a) or (b) above, it is possible to generate empty character strings. For example:

H++ 0L

As address constants, empty character strings are valid and have a zero value. As literals, they are illegal and produce an error. As an item in DATA or LIT, they are legal and produce no values. In LIT, however, one or more of the items listed must be non-empty.

Examples of Character Data Use	Data Produced (octal)
SA1 X3+3RCIO	5213031117
SB6 X0+1L$	6260530000
VFD 30/0HIOTA,6/1RA,24/0AX+1	11172401550155555531
SA1 =H+LEFTΔJUSTIFYΔWITHΔBLANKS+	14050624551225232411
	06315527112410550214
	01161323555555555555
SA1 =0CTENCHARCTS	24051603100122032423
	00000000000000000000
LIT RA+-*/(A,6L)$=Δ,.,0C0,0L,20HLITERALS	
	00000000004546475051
	52535455565700000000
	33000000000000000000
	14112405220114235555
	55555555555555555555
DATA L*ERRORΔINΔPDQΔ*,15B,10HΔΔΔΔΔΔΔΔΔΔ	
	05222217225511165520
	04215500000000000000
	00000000000000000015
	55555555555555555555
SX3 1R → . +1	7130000060
VFD 42/0LOUTPUT,18/1	17252420252400000001

3.7.2 NUMERIC DATA

Numeric data items define values. A data item may consist of the following parts:

	Specified with			
(1) Sign	e	+	-	
(2) Pre-radix	O	D	B	e
(3) Integer	e	n		
(4) Fraction	.e	.n		
(5) Scale(base 10)-single precision	E	En	E±n	
(6) Scale(base 10)-double precision	EE	EEn	EE±n	
(7) Binary scale (base 2)	S	Sn	S±n	
(8) Binary point position	P	Pn	P±n	
(9) Post-radix	O	D	B	e

e indicates empty or not present; n is a numeric string

(1) Sign: + or - may appear as the first character of a data item; if no sign is present, + is assumed.

(2) Pre-radix: Alternative to post-radix. D indicates the value section is expressed in decimal notation; B or O indicates octal notation. The radix pertains only to the value section — integer and fraction. Only one radix specification may be included in a numeric data item.

(3,4) Value Section: A string of digits identifies an integer value; a decimal point identifies a floating point value. When the radix is octal, neither (8) or (9) may appear in the value section.

The value section may contain no more than 32 significant digits if octal or not exceed 7.9×10^{28}. Extra significant digits may cause erroneous results.

The modifier section is part of the value section. The modifiers (E or EE, S, P, post-radix) may appear in any order, but a given modifier may appear only once.

(5,6) Decimal Scale: A modifier of the form E+n or EE+n defines a power of 10 scale factor. E denotes a single precision value; EE a double precision value. The sign is optional; if omitted, + is assumed. The scale value is a decimal integer (regardless of the nominal base). The effect of this scale is to multiply the number by 10 raised to the specified value. The scale value must not exceed +32767. Both fixed and floating point numbers may be scaled. If the scale specifier EE is used with a fixed point number, it still produces a fixed point number in single precision.

(7) Binary Scale: A modifier of the form S±n defines a power of 2 scale factor. The sign is optional; if omitted, + is assumed. The scale value is a decimal integer. The effect of this scale is to multiply the number by 2 raised to the specified value. The scale value must not exceed 32767 in absolute value. Both fixed and floating point numbers may be binary scaled.

(8) Binary Point Position: A modifier of the form P±n places the binary point in a floating point number to represent an unnormalized floating point number. The sign is optional if omitted, + is assumed. Placing the binary point is equivalent to fixing the exponent.

With a P scale, the exponent is adjusted to a value of -(P scale factor). Thus, a number with P-6 will have a biased exponent of 2006_8, and P10 will have an exponent of 1765_8. The value is shifted accordingly.

Another way of explaining P scale:

The number is aligned so that the binary point occurs to the right of the n^{th} bit (counting from low order). The exponent will be adjusted accordingly. Thus a P0 number is an unnormalized integer in floating point notation.

P scales may be specified only for floating point numbers of single or double precision. To avoid an error indication, the high order significant bit must be within the fraction portion of the number.

(9) Radix: D, O, or B defines the radix of the value section. D defines radix 10; O or B defines radix 8. Either a pre-radix or a post-radix may appear, not both. When radix is not specified, the base of the number is derived from the BASE pseudo instruction.

The valid ranges for numbers are restricted by the hardware, although scale factors may exceed valid ranges:

The number 1.0E400S-1200 yields a number which is approximately 5.8×10^{38} and is in range of the floating point representation.

All scaling calculations are performed in 144-bit precision and rounded to 96-bit precision. For single precision, addition rounding is performed to yield 48-bit precision.

In PP assemblies, only fixed point values are permitted.

Examples of numeric data (assume decimal radix):

7	0000	0000	0000	0000	0007
-9	7777	7777	7777	7777	7766
+B13	0000	0000	0000	0000	0013
14BS1	0000	0000	0000	0000	0030
24BE-1	0000	0000	0000	0000	0002
1.0	1720	4000	0000	0000	0000
1.0EE1	1723	5000	0000	0000	0000
	1643	0000	0000	0000	0000
1.0E+1P0	2000	0000	0000	0000	0012
3.2P1S-5E1	1776	0000	0000	0000	0002
0.0151E+01	1715	4651	7676	3554	4264
0.1P47	1720	0314	6314	6314	6314
-D19	7777	7777	7777	7777	7754
-E	7777	7777	7777	7777	7777
DEES	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000

3.8 LITERALS

A literal may be defined as a read-only constant. A literal is stored by the assembler at the end of the zero block, and the address of that data is substituted in the instruction referring to the literal. The process eliminates duplication of read-only data item values and obviates searching for duplicate values.

In an address expression, literals are specified by =n where n is a character or numeric data item. At the first appearance of a value in a literal, COMPASS enters that value into a literal table. Contents of the table entry are used when a subsequent literal refers to that particular value.

Example:

```
SB2    =1
SB3    =1RA
SB4    =2
```

The first statement creates a word in the literal table containing the value 00000000000000000001. The address of that entry is then used in the address field of both statements SB2 and SB3. (The literal in statement SB3 specifies a right justified character A, which also has the value 1.) The SB4 statement creates an entry in the literal table with the value 00000000000000000002, and the address of that entry is in the address field of statement SB4.

COMPASS also permits symbolic reference to literal table entries. Data values can be entered into the literal table and symbols associated with them through the LIT pseudo instruction. Then these entries may be symbolically referenced. The following code sequence will produce the same results as the example above:

```
A    LIT    1,2
     SB2    A
     SB3    A
     SB4    A+1
```

Data items in a LIT variable field always appear in the literal table in the order listed. Literal data values may be character or numeric and are specified just like data items, as follows:

Type	Format
Character	
Delimited by subfield end	=0 [C/H/A/R/L] ⟨character string not including blank , ; → for data items or + - , * / ; → or blank for adress constants⟩
Delimited by character count	=n [C/H/A/R/L] ⟨any characters not including ; →⟩
Delimited by delimiter	= [C/H/A/R/L] d ⟨any character string not including d ; → or ≠⟩ d

Type	Format
Numeric	
Octal	=Bnumeric
Octal	=Onumeric
Decimal	=Dnumeric
Octal	=numericB
Octal	=numericO
Decimal	=numericD

numeric may be an integer, fixed point, or floating point data item, and + or - may immediately follow =. If no B, O, or D appears, the base is assumed to be whatever is currently in use.

3.9 CONSTANTS

A constant is a string of characters which specifies an octal, decimal, or character value. Constants may be used in address expressions of machine and pseudo instructions. Size of a constant depends on the subfield size.

To be recognized as a constant, an item must begin with a numeric character; otherwise it follows the rules for data items. If B, O, or D is not specified, the base is assumed to be whatever is currently in use.

Example of address constants:

```
    SA1  X1+1R
XY  EQU  3HXXX
    VFD  60/0RMESSAGE,30/3LCIO,30/0R0
    SA2  0L(Z)
```

3.10 SPECIAL ELEMENTS

The character * represents the value of the location counter at the beginning of the field. The characters *L and * are equivalent.

The character *O represents the current value of the origin counter.

The character $ represents the position counter value. In an instruction which does not generate code, such as a conditional, the value of $ is usually 59, 44, 29, or 14 in CP assembly, or 11 in PP assembly; however it may be another value if the previous instruction in the block was a VFD. $ reflects next available bit position and is one less than the number of bits still available in a word.

3.11 ADDRESS EXPRESSIONS

An address expression may appear as a subfield in the variable field of a machine or pseudo instruction. An address expression consists of terms joined by the operators + and -. A term consists of elements joined by the operators * and /.

An element, the basic component of an address expression, is one of the following:

symbol

constant

special element: *, *L, *O, or $

deferred symbol: =Ssymbol or =Xsymbol

A term is a combination of elements and a term operator * (multiplication) or / (division). A term must begin with an element and may consists of any number of elements joined by * or /. Two successive elements are illegal. However, ** is permitted since only one of the asterisks is considered an operator. The last element in a term may be omitted; COMPASS then provides an element with zero value.

Examples:

```
A         A*B       72**
*L/2      $
```

3.11.1 EVALUATION OF EXPRESSIONS

An expression is composed of a single term or a number of terms joined by the additive operators + and -. If two or more of the additive operators appear together, a term with a zero value between them is assumed.

A literal (=n) may be used as a term only if it is the last term in the expression. (This avoids confusion regarding the use of +n at the end of a literal.)

Examples:

```
A                +-A*A       A*B-72**
$-29             1+=1        *+3
1+=3.14159EE
```

When an expression is evaluated, each element is replaced with its 60-bit value: constants are replaced with their values; and for address elements, which are 21-bit quantities (literals, *, symbols), the signs are extended to 60 bits.

Within a term, calculation is performed from left to right according to the following rules:

In division, the integral part of the quotient is retained and any remainder is discarded; thus, 5/2*2 results in 4.

Division by zero results in zero and no error.

Only one relocatable or external element may be used in a term; thus **A is illegal in a relocatable assembly if A and * are relocatable.

To the left of a division (divisor), only absolute values may appear.

After terms are evaluated, they are combined, left to right, into an expression. As a result of calculation, only the following forms are permitted:

Absolute value

External value ± constant

± Relocatable ± constant

Terms may cancel relocation values. For example, if A, B, and C are defined as program relocatable symbols, relative to the same base, 3*A-B-C is a permissible expression resulting in single program relocation.

OPERATION CODES 4

4.1 CENTRAL PROCESSOR INSTRUCTIONS

Instructions are either 15 or 30 bits in length, except XJ which is 60 bits. Both formats use a 6-bit operation code and 3-bit result register. The number of bits used for the operand varies with the instruction.

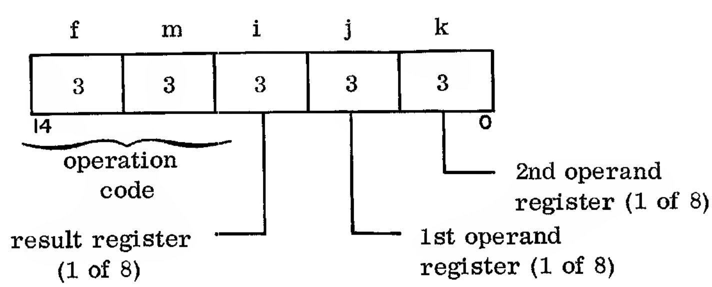
4.1.1 INSTRUCTION FORMATS

The parameters used in the instructions are defined as follows:

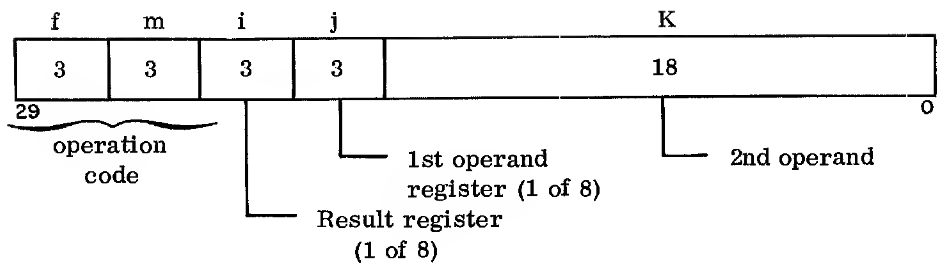
fm	Operation code (6 bits)
i	Result register or X register condition for a branch (3 bits)
j	First operand register (3 bits)
k	Second operand register (3 bits)
jk	Constant, indicating number of shifts (6 bits)
K	Constant, indicating branch destination or second operand (18 bits)
A	One of eight 18-bit address registers
B	One of eight 18-bit increment registers
X	One of eight 60-bit operand registers

The instruction formats are as follows:

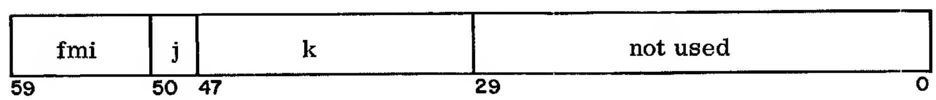
- For a 15-bit instruction:



- For a 30-bit instruction:



- For a 60-bit instruction:
  (applies to Central Exchange Jump only)



4.1.2 OPERATING REGISTERS

The 24 operating registers are identified by letters and digits:

A0,A1,...A7	Address registers
B0,B1,...B7	Increment registers
X0,X1,...X7	Operand registers

<u>A Register</u>

Execution of the SAi (i = 1-5) instruction produces an immediate memory reference to the address contained in Ai and reads the contents of that location into the corresponding operand register Xi (i = 1-5). When SAi (i = 6 or 7) is executed, contents of the corresponding X register are stored at the address specified by Ai. The address register A0 is used for temporary storage; execution of SA0 does not affect X0.

Examples:

SA3	A4+10	Adds 10 to the address in A4 and sets the A3 register to this sum. The X3 register is set to the contents of the location specified by the new A3.
SA6	A2-15	Stores the contents of X6 into the location obtained by subtracting 15 from the address in A2.

B Register

The increment register B0 is set permanently to an 18-bit positive zero which is used to compare for a zero value as an unconditional jump modifier. B1-B7 are used for modifying and program indexing.

The modifying function of the B register is demonstrated by the following example:

SB3	B5+B4	Adds the values contained in the two increment registers, B5 and B4, and places the result in B3.

Example of B register used as an index register:

SA1	ALFA+B3	Sets A1 to the value ALFA plus the contents of B3.
JP	LOC+B6	Causes a program jump to LOC modified by the contents of B6.

X Register

Any of the registers X0-X7 may be used as a result or operand register. X1-X5 hold operands read from central memory; X6 and X7 hold results sent to central memory. The operand registers may be used and changed without causing a change in the corresponding address registers.

Examples:

BX2	X2+X4	Performs the logical addition of X2 and X4 and places the resultant sum in X2.
SX6	A2-B5	Subtracts the contents of B5 from the contents of A2 and stores the result in X6.

4.1.3 EXECUTION

Execution times for instructions are listed in Appendix G. Execution times include readying the next instruction.

6600

After an exchange jump start by a PP and CP program, CP instructions are sent automatically, in the original sequence, to an instruction stack, which holds up to 32 instructions.

Instructions are read from the stack one at a time and issued to the functional units for execution. A scoreboard reservation system in CP control keeps a current log of which units and operating registers are reserved for computation results from functional units.

Each unit executes several instructions, but only one at a time. Some branch instructions require two units, the second unit receives direction from the branch unit.

The rate of issuing instructions may vary from a theoretical maximum of one instruction every 100 nanoseconds (one minor cycle). Sustained issuing at this rate may not be possible because of unit and CM conflict or because of serial rather than simultaneous operation of units. Program running time can be decreased by efficient use of the units. Instructions that are not dependent on previous steps may be arranged or nested in program areas where they may be executed concurrently with other operations to eliminate dead spots in the program and increase the instruction issue rate.

The following steps summarize instruction issuing and execution:

- An instruction is issued to a function unit when:

    Specified functional unit is not reserved

    Specified result register is not reserved for a previous result

- Instructions are issued to functional units at minor cycle intervals when no reservation conflicts are present.

- Instruction execution starts in a functional unit when both operands are available. Execution is delayed when an operand is a result of a previous step which is not complete.

- No delay occurs between the end of a first unit and the start of a second unit which is waiting for the results of the first.

- After a branch instruction no further instructions are issued until instruction has been executed. In the execution of a branch instruction, the branch unit uses:

    Increment unit to form the instructions GO TO K + Bi and GO TO K if Bi . . .

    Long add unit to perform the instruction GO TO K if Xj . . .

  Time spent in the long add or increment units is part of total braneh time.

Read central memory access time is computed from the end of increment unit time to the time an operand is available in X operand register. Minimum time is 500 nanoseconds assuming no central memory bank conflict.

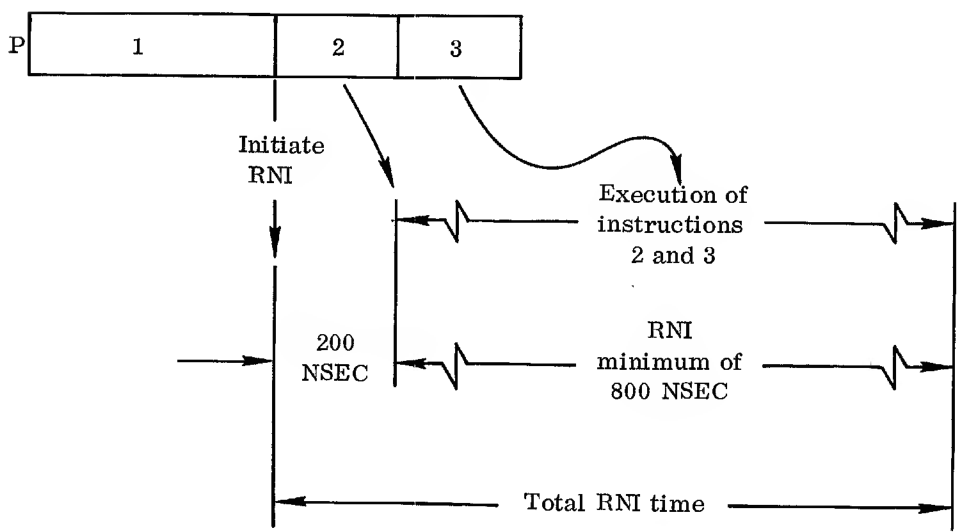
6400/6500

The 6400 and 6500 systems CP has a unified arithmetic unit, rather than separate functional units as in the 6600 system. Instructions in the 6400 and 6500 CP are executed sequentially.

For efficient coding in the 6400 and the 6500 central processor:

> Always attempt to place jump instructions in the upper portion of the instruction word to avoid both the additional time for RNI (2 minor cycles) and the possibility of a memory bank conflict with (P + 1).
>
> Where possible, place load/store instructions in the lower two portions to avoid lengthening execution times.

Reading the next instruction words of a program from central memory, RNI, is partially concurrent with instruction execution. RNI is initiated between execution of the first and second instructions of the word being processed. Initiating RNI operation requires two minor cycles; the remainder of the RNI is parallel in time with execution of the remaining instructions in the word:



In calculating execution times, two minor cycles are added to each instruction word in a program to cover the RNI initiation time. Exceptions are the return jump and the jump instructions (in which the jump condition is met) when they occupy the upper position of the instruction word. Since the times for these instructions already include the time required to read the new instruction word at the jump address, no additional time is consumed (Appendix G).

Example:

P	Jump to K (met)		Pass	Pass
K	Add 1	Add 2	Load	Load

<u>Instruction</u>	<u>Minor Cycles Required</u>
Jump	13
Add 1	5
RNI Initiation	2
Add 2	5
Load	12
Store	10
Total Time	47 Minor Cycles

After RNI is initiated (between the first and second instructions of the word), a minimum of eight minor cycles elapses before the next instruction word is available for execution. Even if the lower order positions of the word should require less than eight minor cycles, a minimum of eight minor cycles is allowed regardless of the execution times stated in Appendix G.

Example:

P	Jump to K (not met)	Pass	Pass
P + 1			

Instruction	Minor Cycles Required
Jump (not met)	5
RNI Initiation	2
Pass=3, Pass=3 } -RNI minimum	8
Minimum time before word at P+1 is available for execution	15

The return jump instruction, all jump instructions in which the jump condition is met, and load/store memory instructions require additional time when they are in the second position of an instruction word. This additional time requirement results from hardware limitations rather than memory bank conflicts.

Instruction	Additional Time for Second Instruction in Word
Jumps 02-07 (jump condition met)	1 minor cycle
Return Jump 010	2 minor cycles
Load/Store (5X with i $\neq$ 0)	2 minor cycles

If the second instruction of a word references the memory bank containing (P+1), a bank conflict requires an additional three minor cycles.

If a store (not load) as the first instruction of a word causes a bank conflict with (P+1), three minor cycles are added to the execution time.

4.1.4 OPERATION CODES

Instructions for the central processor are listed below; they are arranged by unit function and mnemonic code. Each mnemonic is followed by a format model, a mnemonic description, the instruction bit size in parentheses and the octal code. In the examples, K represents a variable which may be coded as one of the following:

- One or more decimal or octal integers, symbolic constants, or ordinary symbols, connected by operators
- External symbol
- Common block segment name, alone or followed by a plus sign and an integer or symbolic constant
- Literal

Subfields within the variable field may appear in any order.

NO	No operation (pass)	(15)	46000

A do-nothing instruction used typically to pad between program steps. A comment on the same card should begin with a period; otherwise it will appear to be an address field and may cause an error flag.

INCREMENT UNIT

Performs one's complement addition and subtraction of 18-bit numbers. The following instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in Ai.

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the K portion of the instruction. K is an 18-bit signed constant. If the sign of K is minus in instructions 50xxx, 51xxx, and 52xxx, the 18-bit one's complement of K is placed in the K portion of the instruction word. Operands obtained from an X register are the truncated lower 18 bits of the 60-bit register. The operands may appear in any order.

An immediate memory reference to the address specified by the final contents of address register Ai is effected by the execution of a SAi (i = 1-7) instruction. The operand read from memory address specified by A1-A5 is sent to the corresponding operand register X1-X5. The operand from X6 or X7 is stored at the address specified by the corresponding A6 or A7. There is no corresponding relation between the A0 and X0 registers.

SAi Aj±K	Sum/difference Aj±K to Ai	(30)	50ijk

Examples:

SA2 A2+K	Adds K value to contents of A2 register and places result in A2 register. X2 register contains the contents of address referenced in A2 register.
SA7 A2+K	Adds K value to contents of A2 register and places result in A7 register. Contents of X7 register are stored at address referenced in A7 register.
SA2 K+A2	
SA2 A3-K	

SAi K	(K+B0 to Ai)	(30)	51i0k
SAi Bj±K	Sum/difference Bj±K to Ai	(30)	51ijk

Examples:

SA2 B3+K — Adds increment value in B3 register to K value and places result in A2 register. X2 register contains the contents of address referenced in A2 register.
SA2 K+B3
SA2 B2-K
SA2 K

SAi Xj±K	Sum/difference Xj±K to Ai	(30)	52ijk

Examples:

SA2 X3+K — Adds lower 18 bits (only) of X3 register to value of K and places result in A2 register. X2 register contains the contents of address referenced in A2 register.
SA2 K+X3
SA2 X3-K

SAi Xj	(Xj+B0 to Ai)	(15)	53ij0
SAi Xj+Bk	Sum Xj+Bk to Ai	(15)	53ijk

Examples:

SA2 X3+B4
SA2 B4+X3
SA2 X3

SAi Aj	(Aj+B0 to Ai)	(15)	54ij0
SAi Aj+Bk	Sum Aj+Bk to Ai	(15)	54ijk

Examples:

SA2 A3+B4
SA2 B4+A3
SA2 A3

SAi Aj-Bk	Difference Aj-Bk to Ai Examples: SA2 A3-B4 SA2 -B4+A3	(15)	55ijk
SAi Bj	(Bj+B0 to Ai)	(15)	56ij0
SAi Bj+Bk	Sum Bj+Bk to Ai Examples: SA2 B3+B4 SA2 B3	(15)	56ijk
SAi -Bk	(-Bk+B0 to Ai)	(15)	57i0k
SAi Bj-Bk	Difference Bj-Bk to Ai Examples: SA2 B3-B4 SA2 -B4+B3 SA2 -B4	(15)	57ijk

The following instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in Bi.

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the K portion of the instruction. K is an 18-bit signed constant. If the sign of K is minus in instructions 60xxx, 61xxx and 62xxx, the 18-bit one's complement of K is placed in the K portion of the instruction word. Operands obtained from an X register are the truncated lower 18 bits of the 60-bit register.

The operands may appear in any order and are formatted in the same manner as the parallel SAi instructions.

SBi Aj±K	Sum/difference Aj±K to Bi	(30)	60ijk
SBi K	Value of K (K+B0) to Bi	(30)	61i0k
SBi Bj±K	Sum/difference Bj±K to Bi	(30)	61ijk
SBi Xj±K	Sum/difference Xj±K to Bi	(30)	62ijk
SBi Xj	Value of Xj (Xj+B0) to Bi	(15)	63ij0
SBi Xj+Bk	Sum Xj+Bk to Bi	(15)	63ijk

SBi Aj	Value of Aj (Aj+B0) to Bi	(15)	64ij0
SBi Aj+Bk	Sum Aj+Bk to Bi	(15)	64ijk
SBi Aj-Bk	Difference Aj-Bk to Bi	(15)	65ijk
SBi Bj	Value of Bj (Bj+B0) to Bi	(15)	66ij0
SBi Bj+Bk	Sum Bj+Bk to Bi	(15)	66ijk
SBi -Bk	Value of -Bk (-Bk+B0) to Bi	(15)	67i0k
SBi Bj-Bk	Difference Bj-Bk to Bi	(15)	67ijk

The following instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in Xi. (Boolean instructions must be used to perform arithmetic operations on 60-bit operands.)

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the K portion of the instruction. K is an 18-bit signed constant. If the sign of K is minus in instructions 70xxx, 71xxx and 72xxx, the 18-bit one's complement of K is placed in the K portion of the instruction word.

Operands obtained from an Xj register are the truncated lower 18 bits of the 60-bit register. Conversely, an 18-bit result placed in Xi carries the sign bit extended to the remaining bits of the 60-bit register.

The operands may appear in any order and are formatted in the same manner as the parallel SAi instructions.

SXi Aj±K	Sum/difference Aj±K to Xi	(30)	70ijk
SXi K	Value of K (K+B0) to Xi	(30)	71i0k
SXi Bj±K	Sum/difference Bj±K to Xi	(30)	71ijk
SXi Xj±K	Sum/difference Xj±K to Xi	(30)	72ijk
SXi Xj	Value of Xj (Xj+B0) to Xi	(15)	73ij0
SXi Xj+Bk	Sum Xj+Bk to Xi	(15)	73ijk
SXi Aj	Value of Aj (Aj+B0) to Xi	(15)	74ij0
SXi Aj+Bk	Sum Aj+Bk to Xi	(15)	74ijk

SXi Aj-Bk	Difference Aj-Bk to Xi	(15)	75ijk
SXi Bj	Value of Bj (Bj+B0) to Xi	(15)	76ij0
SXi -Bj	Value of -Bj (-Bj+B0) to Xi	(15)	76i0k
SXi Bj+Bk	Sum Bj+Bk to Xi	(15)	76ijk
SXi Bj-Bk	Difference Bj-Bk to Xi	(15)	77ijk

BRANCH UNIT — Handles all jumps or branches from the program.

PS — Program stop. (30) 0000000000

Stops the CP at the current instruction. An exchange jump is necessary to restart the CP. The program stop instruction is forced upper and forces the next instruction upper.

RJ K — Return jump to K. (30) 0100k

Stores an unconditional jump (0400) and the current program address plus one in the upper 30 bits of K and then branches to K+1 for the next instruction. As a result the contents of K appear as follows:

```
EQ  B0, B0, L+1
PS
```

where L is the address of the executed RJ instruction.

XJ Bj+K — Central exchange jump to K (60) 0130000000 4600046000

Unconditionally exchange jumps to the CP, regardless of the state of the monitor flag bit. Depending on whether the monitor flag bit is set or clear, operation is as follows:

If the monitor flag bit is clear, the starting address (absolute) for the exchange is taken from the 18-bit monitor address register. During the exchange, the monitor flag bit is set.

XJ Bj+K (Cont'd)	If the monitor flag bit is set, the starting address (absolute) for the exchange is the 18-bit result formed by adding K to the contents of register Bj. During the exchange, the monitor flag bit is cleared.		
JP Bi+K	Jump to Bi+K	(30)	02i0k
	Adds the contents of Bi to K and branches to the address specified by the sum. When Bi = B0, the branch address is K. Addition is performed modulo $2^{18}-1$.		

The following instructions all branch to K when the word in operand register Xj meets the conditions specified.

ZR Xj,K	Jump to K if Xj = 0	(30)	030jk
	Branches to K if Xj is equal to zero. If the condition is not met, the next consecutive instruction step is executcd. The test is made in the long add unit. Minus zero and plus zero both satisfy the test.		
NZ Xj,K	Jump to K if Xj $\neq$ 0	(30)	031jk
	Branches to K if Xj is not equal to zero. If the condition is not met, the next consecutive instruction step is exeeuted. The test is made in the long add unit. Plus zero and minus zero do not satisfy the test.		
PL Xj,K	Jump to K if Xj is positive	(30)	032jk
	Branches to K if Xj is positive. If the condition is not met, the next consecutive instruction step is executed.		
NG Xj,K	Jump to K if Xj is negative	(30)	033jk
	Branches to K if Xj is negative. If the condition is not met, the next consecutive instruction step is executed.		

IR Xj,K	Jump to K if Xj is in range	(30)	034jk
	Branches to K if Xj is less than infinity ($377700\ldots0_8$).		
OR Xj,K	Jump to K if Xj is out of range	(30)	035jk
	Branches to K if Xj is greater than or equal to $377700\ldots0_8$.		
DF Xj,K	Jump to K if Xj is definite	(30)	036jk
	Branches to K if Xj is definite. The test is a comparison against an indefinite quantity ($177700\ldots0_8$).		
ID Xj,K	Jump to K if Xj is indefinite	(30)	037jk
	Branches to K if Xj is indefinite. The test is a comparison against an indefinite quantity ($177700\ldots0_8$).		

The following instructions all branch to K when the word in register Bi meets the condition specified in register Bj:

ZR K	Jump to K	(30)	0400k
ZR Bi,K	Jump to K if Bi = B0	(30)	04i0k
	Compares Bi with B0 and branches to K if Bi is zero. Minus zero in Bi does not satisfy this test. ZR K,B2 is equivalent to EQ B0,B2,K		
EQ K	Jump to K	(30)	0400k
	EQ K assembles as EQ B0,B0,K an unconditional jump.		
EQ Bi,K	Jump to K if Bi = 0	(30)	04i0k
EQ Bi,Bj,K	Jump to K if Bi = Bj	(30)	04ijk
	Compares Bi with Bj and branches to K if Bi is equal to Bj. Minus zero is not equal to plus zero. EQ Bi,K assembles as EQ Bi,B0,K		

NE Bi,K	Jump to K if Bi $\neq$ 0	(30)	05i0k
NE Bi,Bj,K	Jump to K if Bi $\neq$ Bj Compares Bi with Bj and branches to K if Bi is not equal to Bj. Minus zero is not equal to plus zero. NE Bi,K assembles as NE Bi,B0,K	(30)	05ijk
NZ Bi,K	Jump to K if Bi $\neq$ B0 Compares Bi with B0 and branches to K if Bi is not zero. Minus zero in Bi satisfies this test. NZ K,B2 is equivalent to NE B0,B2,K	(30)	05i0k
LE Bj,K	Jump to K if Bj $\leq$ 0 Compares Bi with B0 and branches to K if result is negative. LE K,B1 is equivalent to LE B1,B0,K	(30)	060jk
PL Bi,K	Jump to K if Bi $\geq$ B0 Compares Bi with B0 and branches to K if the result is positive. PL K,B1 is equivalent to GE B1,B0,K	(30)	06i0k
GE Bi,K	Jump to K if Bi $\geq$ 0	(30)	06ijk
GE Bi,Bj,K	Jump to K if Bi $\geq$ Bj Compares Bi with Bj and branches to K if Bi is greater than or equal to Bj. Plus zero is greater than minus zero.	(30)	06ijk
LE Bj,Bi,K	Jump to K if Bj $\leq$ Bi Compares Bi with Bj and branches to K if Bj is less than or equal to Bi. Plus zero is greater than minus zero.	(30)	06ijk
GT Bj,K	Jump to K if Bj > 0 Compares Bi with B0 and branches to K if the result is greater than 0. GT K,B1 is equivalent to GT B1,B0,K	(30)	070jk
LT Bi,K	Jump to K if Bi < 0 Compares Bi with B0 and branches to K if the result is negative. LT K,B1 is equivalent to LT B1,B0,K	(30)	07i0k

NG Bi,K	Jump to K if Bi < B0	(30)	07i0k

Compares Bi with B0 and branches to K if Bi is negative. NG K,B1 is equivalent to LT Bi,B0,K

GT Bj,Bi,K	Jump to K if Bj > Bi	(30)	07ijk

Compares Bj with Bi and branches to K if Bi is greater than Bj. Plus zero is greater than minus zero.

LT Bi,Bj,K	Jump to K if Bi < Bj	(30)	07ijk

Compares Bi with Bj and branches to K if Bi is less than Bj. Minus zero is less than plus zero.

BOOLEAN UNIT

Handles the basic logical operations of transfer, logical product, logical sum and logical difference.

BXi Xj	Transmit Xj to Xi	(15)	10ijj

Transfers the 60-bit word in operand register Xj to Xi.

BXi Xj*Xk	Logical product of Xj and Xk to Xi	(15)	11ijk

Forms the logical product (AND function) of the 60-bit words in operand registers Xj and Xk and places the result in Xi. (Bits of register Xi are set to 1 when the corresponding bits of the Xj and Xk registers are 1.)

```
Xj  0101
Xk  1100
Xi  0100
```

BXi Xj+Xk	Logical sum of Xj and Xk to Xi	(15)	12ijk

Forms the logical sum (inclusive OR) of the 60-bit words in operand registers Xj and Xk and places the result in Xi. (Bits of register Xi are set to 1 if the corresponding bits of the Xj or Xk register are 1.)

```
Xj  0101
Xk  1100
Xi  1101
```

BXi Xj-Xk	Logical difference of Xj and Xk to Xi	(15)	13ijk

Forms the logical difference (exclusive OR) of the 60-bit words in operand registers Xj and Xk and places the result in Xi. (Bits of register Xi are set to 1 if the corresponding bits in the Xj and Xk registers are unlike.)

```
Xj  0101
Xk  1100
    ----
Xi  1001
```

BXi -Xk	Transmit the complement of Xk to Xi	(15)	14ikk

Extracts the 60-bit word from operand register Xk, complements it, and transmits the complement to operand register Xi. The contents of Xk are not changed.

BXi -Xk*Xj	Logical product of Xj and complement of Xk to Xi	(15)	15ijk

Forms in Xi the logical product (AND function) of Xj and the complement of Xk. Contents of Xk and Xj are not changed.

```
Step 1  Xj  0101    Step 2  Xj   0101
        Xk  1100           -Xk   0011
                                 ----
                            Xi   0001
```

BXi -Xk+Xj	Logical sum of Xj and complement of Xk to Xi	(15)	16ijk

Complements the 60-bit word in Xk, forms the logical sum (inclusive OR) of this quantity and Xj, and places the result in Xi. Contents of Xk and Xj are not changed.

```
Step 1  Xj  0101    Step 2  Xj   0101
        Xk  1100           -Xk   0011
                                 ----
                            Xi   0111
```

BXi -Xk-Xj	Logical difference of Xj and complement of Xk to Xi	(15)	17ijk
	Complements the 60-bit word in Xk, forms the difference (exclusive OR) of this quantity and Xj, and places the result in Xi. Contents of Xk and Xj are not changed.		

```
Step 1  Xj  0101    Step 2  Xj   0101
        Xk  1100            -Xk  0011
                            Xi   0110
```

SHIFT UNIT — Handles shifting operations including left (circular) and right (end-off/sign extension) shift, normalize, pack and unpack floating point operations. The unit provides also a mask generator.

LXi jk	Shift Xi left jk places	(15)	20ijk
	Shifts the 60-bit word in Xi left circular jk places. Each step moves the leftmost bit of Xi into the rightmost position of Xi. The 6-bit shift count jk is coded as an octal or decimal number. A complete circular shift of Xi is possible (jk = 60). Example: LX2 36		
AXi jk	Arithmetic right shift Xi, jk places	(15)	21ijk
	Shifts the 60-bit word in Xi right jk places. The rightmost bits of Xi are discarded and the sign bit is extended. The 6-bit shift count jk is coded as an octal or decimal number. Example: AX2 36		
LXi Xk	Transmit Xk to Xi	(15)	22i0k
	Transfers the 60-bit word in operand register Xk to Xi. Equivalent to the BXi Xj except this instruction executes in the shift unit and is, therefore, preferable if the Boolean unit is busy. The BXi Xj is always preferable in a 6400 or 6500, as it is faster.		

LXi Bj,Xk	Left shift Xk nominally Bj places to Xi	(15)	22ijk

Shifts the 60-bit word in Xk the number of places specified by the low order 6 bits of Bj and places the result in Xi.

If Bj is positive, Xk is shifted left circular.

If Bj is negative, Xk is shifted right (end off with sign extension) and the complement of the low order 6 bits of Bj gives the number of places to be shifted.

Example: LX2 B1,X3

AXi Bj,Xk	Arithmetic right shift Xk nominally Bj places to Xi	(15)	23ijk

Shifts the 60-bit word in Xk the number of places specified by the low order 6 bits of Bj and places the result in Xi.

If Bj is positive, Xk is shifted right (end off with sign extension).

If Bj is negative, Xk is shifted left circular; and the complement of the low order 6 bits of Bj gives the number of places to be shifted.

Example: AX2 B3,X4

NXi Xk	Normalize Xk to Xi	(15)	24i0k
NXi Bj,Xk	Normalize Xk in Xi and Bj	(15)	24ijk

Normalizes the floating point quantity in Xk and places it in Xi. The number of left shifts required is placed in Bj during the operation. If the coefficient of Xk is zero, Xi is cleared to all zeros and Bj is set to 48. If the size of the exponent is less than the number of leading zeros in the coefficient of Xk, underflow occurs during normalizing and the exponent and coefficient of Xi are both cleared.

Example: NX2 B3,X4

ZXi Xk	Round and normalize Xk in Xi	(15)	25i0k
ZXi Bj,Xk	Round and normalize Xk in Xi and Bj	(15)	25ijk

Performs the same operation as NX (24 ijk) except that the quantity in Xk is rounded before it is normalized. Normalizing a zero coefficient places the round bit in bit 47 and reduces the exponent by 48.

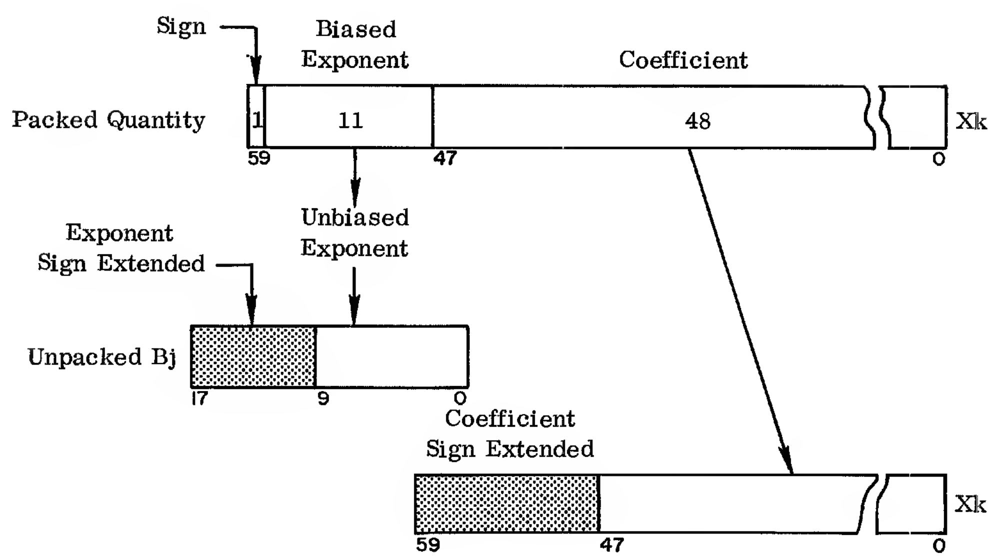
Example: ZX2 B3,X4

UXi Xk	Unpack Xk to Xi	(15)	26i0k
UXi Bj,Xk	Unpack Xk to Xi and Bj	(15)	26ijk

Unpacks the floating point quantity in Xk and sends the sign and 48-bit coefficient to Xi and the 11-bit exponent minus 2000_8 to Bj which then contains the true one's complement representation of the exponent. Xk may be an unnormalized number.

Example: UX2 B3,X4

The exponent and coefficient are sent to the low order bits of the registers as shown in the following diagram.



PXi Bj,Xk	Pack Xi from Xk and Bj	(15)	27ijk

Packs a floating point number in Xi. The coefficient of the number is obtained from the sign and low order 48 bits of Xk and the exponent is obtained by adding 2000_8 to the low order 11 bits of Bj. The coefficient is not normalized.

Exponent and coefficient are obtained from the low order bits of the register and packed as shown in the above diagram. During pack, overflow occurs when Bj is a positive number of more than 10 bits; exit on overflow is optional. Underflow occurs (no exit) when Bj is a negative number of more than 10 bits.

Example: PX2 B3,X4

MXi jk	Form mask in Xi, jk bits	(15)	43ijk

Forms a mask in Xi. The 6-bit quantity jk defines the number of ones in the mask as counted from the highest order bit in Xi.

ADD UNIT

Performs floating point addition and subtraction on floating point numbers or their rounded representation.

FXi Xj+Xk	Floating sum of Xj and Xk to Xi	(15)	30ijk

Forms the sum of the floating point quantities in Xj and Xk and packs the result in Xi. The packed result is the upper half of a double precision sum.

Both arguments are unpacked, and the coefficient of the argument with the smaller exponent is entered into the upper half of a non-programmable 96-bit accumulator. The coefficient is shifted right by the difference of the exponents. The other coefficient is then added into the upper half of the accumulator. If overflow occurs, the sum is shifted right one place, and the exponent of the result is increased by one. The upper half of the accumulator holds the coefficient of

FXi Xj+Xk (Cont'd)	the sum, which is not necessarily normalized. The exponent and upper coefficient are then repacked in Xi. If both exponents are zero and no overflow occurs, the instruction effects an ordinary integer addition.		
FXi Xj-Xk	Floating difference of Xj and Xk to Xi Forms the difference of the floating point quantities in Xj and Xk and packs the result in Xi. Alignment and overflow operations are similar to the floating sum (30ijk) instruction, and the difference is not necessarily normalized. The packed result is the upper half of a double precision difference. An ordinary integer subtraction is performed when the exponents are zero.	(15)	31ijk
DXi Xj+Xk	Floating double precision sum of Xj and Xk to Xi Forms the sum of two floating point numbers as in the floating sum (30ijk) instruction, but packs the lower half of the double precision sum with an exponent 48 less than the exponent of the upper sum.	(15)	32ijk
DXi Xj-Xk	Floating double precision difference of Xj and Xk to Xi Forms the difference of two floating point numbers as in the floating difference (31ijk) instruction, but packs the lower half of the double precision difference with an exponent of 48 less than the exponent of the upper difference.	(15)	33ijk
RXi Xj+Xk	Round floating sum of Xj and Xk to Xi Forms the round sum of the floating point quantities in Xj and Xk and packs the upper sum of the double precision result in Xi.	(15)	34ijk

RXi Xj+Xk (Cont'd)	The sum is formed in the same manner as the floating sum (30ijk) instruction except that the operands are rounded before the addition to produce a round sum. If both operands are normalized or the operands have unlike signs, a round bit is attached at the right end of both operands. Otherwise, a round bit is attached at the right end of the operand having the larger exponent.		
RXi Xj-Xk	Round floating difference of Xj and Xk to Xi	(15)	35ijk
	Forms the round difference of the floating point quantities in Xj and Xk and packs the upper difference of the double precision result in Xi. The difference is formed in the same manner as the floating difference (31ijk) instruction except that the operands are rounded before subtraction to produce a round difference. If both operands are normalized or the operands have like signs, a round bit is attached at the right end of both operands; otherwise, a round bit is attached at the right of the operand with the larger exponent.		

LONG ADD UNIT

Performs one's complement addition and subtraction of 60-bit fixed point numbers.

IXi Xj+Xk	Integer sum of Xj and Xk to Xi	(15)	36ijk
	Forms a 60-bit one's complement sum of the quantities in Xj and Xk and stores the result in Xi. An overflow condition is ignored.		
IXi Xj-Xk	Integer difference of Xj and Xk to Xi	(15)	37ijk
	Forms the 60-bit one's complement difference of the quantities in Xj (minuend) and Xk (subtrahend) and stores the result in Xi.		

MULTIPLY UNIT Performs floating point multiplication on floating point numbers or their rounded representations.

FXi Xj*Xk	Floating product of Xj and Xk to Xi	(15)	40ijk
	Multiples the floating point quantities in Xj (multiplier) and Xk (multiplicand) and packs the upper product result in Xi. The result is a normalized quantity only when both operands are normalized; the exponent is then the sum of the exponents plus 47 (or 48). The result is unnormalized when either or both operands are unnormalized; the exponent is then the sum of the exponents plus 48.		
RXi Xj*Xk	Round floating product of Xj and Xk to Xi	(15)	41ijk
	Attaches a round bit to the floating point number in Xk (multiplicand), multiplies this number by the floating point number in Xj, and packs the upper product result in Xi. (No lower product is available.) The result is a normalized quantity only when both operands are normalized; the exponent is then the sum of the exponents plus 47 (or 48). The result is unnormalized when either or both operands are unnormalized; the exponent is then the sum of the exponents plus 48.		
DXi Xj*Xk	Floating double precision product of Xj and Xk to Xi	(15)	42ijk
	Multiplies the floating point quantities in Xj and Xk and packs the lower product in Xi with an exponent 48 less then the exponent of the upper product. The result is not necessarily normalized.		

DIVIDE UNIT Performs floating point division of floating point quantities or their rounded representation. Also, sums the number of ones in a 60-bit word.

FXi Xj/Xk	Floating divide Xj by Xk to Xi	(15)	44ijk
	Divides the floating point quantities in Xj (dividend) by Xk (divisor) and packs the quotient in Xi. The exponent of the result in a no-overflow case is the difference of Xj and Xk exponents minus 48. A one-bit overflow is compensated by shifting the coefficient right one place and increasing the exponent by one. The exponent is then the difference of the Xj and Xk exponents minus 47. The result is a normalized quantity when both Xj and Xk are normalized.		
RXi Xj/Xk	Round floating divide Xj by Xk to Xi	(15)	45ijk
	Divides the floating point quantity in Xj (dividend) by Xk (divisor) and packs the round quotient in Xi. A 1/3 round bit is added to the least significant bit of the dividend (Xj) before division starts. The result exponent in a no-overflow case is the difference of Xj and Xk exponents minus 48. A one-bit overflow is compensated by shifting the coefficient right one place and increasing the exponent by one. The exponent is then the difference of Xj and Xk exponents minus 47. The result is a normalized quantity when both Xj and Xk are normalized.		
CXi Xk	Count the number of ones in Xk to Xi	(15)	47ikk
	Counts the number of ones in Xk and stores the count in the lower order 6 bits of Xi. Bits 6 through 59 are cleared to zero.		

EXTENDED CORE STORAGE UNIT

Provides communication with extended core storage (ECS).

RE Bj+K	Read extended core storage	(30)	011jk
	Initiates a read operation to transfer [(Bj)+K] 60-bit words from ECS to CM. The initial ECS address is [(X0)+RA ECS]; the initial CM address is [(A0)+RA CM]. This instruction must be located in the upper position of the instruction word.		

WE Bj+K	Write extended core storage	(30)	012jk
	Initiates a write operation to transfer [(Bj)+K] 60-bit words from CM to ECS. The initial CM address is [(A0)+RA CM]; the initial ECS address is [(X0)+RA ECS]. This instruction must be located in the upper order position of the instruction word.		

The lower order 30 bits of the instruction word containing the ECS read or write instruction is an error exit and should always hold a jump to an error routine. Two conditions cause an error exit:

- Parity error when reading ECS. If a parity error is detected, the entire block of data is transferred before the exit is taken.
- The ECS bank from/to which data is to be transferred is not available because the bank is in maintenance mode, or the bank is not present in the system.

When either condition exists and an attempt is made to perform a write operation, no data transfer occurs. If the operation is a read and addresses are in range, zeros are transferred to CM.

If an exchange jump occurs while an ECS transfer is in progress, the exchange waits until completion of a record. If the record just completed is the last record of the block transfer, and the transfer was error-free, the CP exits to (P)+1, and the exchange jump takes place; however if an error condition exists, the CP exits to the lower instruction, executes it, and then exchange jump is performed. If the record just completed does not complete the block transfer, the exchange jump occurs, and the contents of P are stored in the exchange jump package. A return exchange jump to this program begins execution with the ECS read or write instruction and restarts the transfer. The transfer does not resume at the point it was truncated; rather, the entire transfer must be repeated.

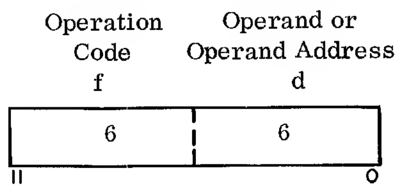
4.2 PERIPHERAL PROCESSOR INSTRUCTIONS

The ten PP processors can communicate with each other and exchange data with CM. Generally, the processors are not used for solving complex arithmetic and logical problems; they are used to perform input/output operations for CP programs, to organize problem data, and to store it in CM. All activity with input/output equipment is directed by PP input/output instructions.

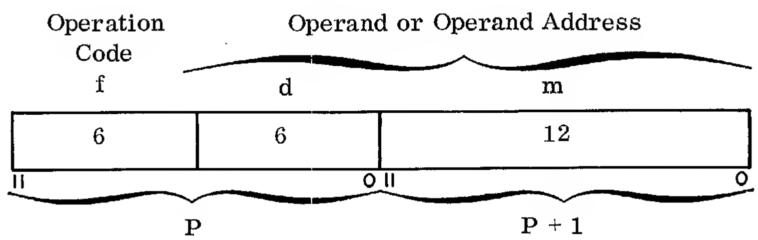
4.2.1 INSTRUCTION FORMAT

A PP instruction may be 12 or 24 bits; a 12-bit PP instruction accommodates a 6-bit or 18-bit operand or operand address; a 24-bit PP instruction accommodates a 6-bit, 12-bit or 18-bit address.

The 12-bit format has a 6-bit operation code and a 6-bit operand or operand address.



The 24-bit format requires two memory words. The 6-bit quantity, d, of the first word is used with the 12-bit quantity, m, of the next consecutive word to form an 18-bit operand or operand address, c.



4.2.2 ADDRESS MODES

Program indexing is accomplished and operands manipulated in three modes:

No address

d or dm is an operand

d = 12-bit number (upper 6 bits = 0)

dm = 18-bit number

Direct address

d or m plus contents of d is the address of an operand

d = address in memory locations $0000\text{-}0077_8$

m + (d) = 12-bit address referencing all possible peripheral memory locations. If d $\neq$ 0, d + m is the operand address; if d = 0, m is the operand address. Thus, location d may be used for an index quantity to modify operand addresses (direct index addressing).

Indirect address

d = address containing the address of the operand

4.2.3 OPERATION CODES

The instructions for the PPs are listed below. They are arranged by unit function and mnemonic code. Each mnemonic code is followed by a variable field description. Subfields are separated by commas. A mnemonic description, the instruction bit size in parentheses and the octal code are shown. The variable subfield symbols are:

d	Index location, 6 bits
m	Address value, 12 bits
c	Address value, 18 bits
r	Numeric value in jump instructions to indicate number of steps to jump

In the variable field, in parentheses indicate the contents of a register or location. Double parentheses indicate indirect addressing. M = indexed direct address (m+(d)).

NO OPERATION CODE

The following instruction specifies that no operation be performed; it provides a means of padding out a program.

PSN	Pass.	(12)	2400

DATA TRANSMISSION CODES

LDN d	Load d Clears the arithmetic (A) register and loads d into the lower 6 bits of A. The upper 12 bits of A are zero.	(12)	14dd
LCN d	Load complement d Clears the A register and loads the complement of d into the lower 6 bits of A. The upper 12 bits of A are set to ones.	(12)	15dd
LDD d	Load (d) Clears the A register and loads the contents of location d into the lower 12 bits of A. The upper 6 bits of A are zero.	(12)	30dd

	STD d	Store (d)	(12)	34dd
		Stores the lower 12 bits of the A register into location d. The contents of A are not altered.		
	LDI d	Load ((d))	(12)	40dd
		Clears the A register and loads into A the 12-bit quantity obtained by indirect addressing. The upper 6 bits of A are zero. Location d is read out of memory, and the word obtained is used as the operand address.		
	STI d	Store ((d))	(12)	44dd
		Stores the lower 12 bits of the A register into the location specified by the contents of d. The contents of A are not altered.		
	LDC c	Load c	(24)	20cc cccc
		Clears the A register and loads the 18-bit quantity consisting of d as the upper 6 bits and m as the lower 12 bits.		
	LDM m,d	Load M	(24)	50dd mmmm
		Clears the A register and loads the 12-bit operand obtained by indexed direct addressing (m+(d)) into the lower 12 bits of A. The upper 6 bits of A are zero. The quantity m, put into memory location P+1, is read out of P+1 and serves as the base operand address to which d is added.		
	STM m,d	Store M	(24)	54dd mmmm
		Stores the lower 12 bits of the A register in the location determined by indexed direct addressing. The contents of A are not altered.		
ARITHMETIC CODES	ADN d	Add d	(12)	16dd
		Adds the 6-bit positive quantity d to the contents of the A register.		
	SBN d	Subtract d	(12)	17dd
		Subtracts the 6-bit positive quantity d from the contents of the A register.		

	ADD d	Add (d) Adds to the A register the 12-bit positive quantity in location d.	(12)	31dd
	SBD d	Subtract (d) Subtracts from the A register the 12-bit positive quantity in location d.	(12)	32dd
	ADI d	Add ((d)) Adds to the contents of the A register a 12-bit positive operand obtained by indirect addressing. Location d is read out of memory and the word obtained is used as the operand address.	(12)	41dd
	SBI d	Subtract ((d)) Subtracts from the A register a 12-bit positive operand obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.	(12)	42dd
	ADC c	Add c Adds to the A register the 18-bit quantity c.	(24)	21cc cccc
	ADM m,d	Add M Adds to the contents of the A register a 12-bit positive operand obtained by indexed direct addressing.	(24)	51dd mmmm
	SBM m,d	Subtract M Subtracts from the A register a 12-bit positive operand obtained by indexed direct addressing.	(24)	52dd mmmm
SHIFT CODE	SHN r	Shift r Shifts contents of A register right or left r places. If r is positive ($00\text{-}37_8$), shift is left circular; if r is negative ($40\text{-}77_8$), A is shifted right (end off with no sign extension). A left shift of 6 places results when r = 6 and a right shift of 6 places results when $r = 71_8$.	(12)	10rr

LOGICAL CODES

LMN d — Logical difference d — (12) — 11dd

Forms in the A register the bit-by-bit logical difference of d and the lower 6 bits of A. Equivalent to complementing the individual bits in A which correspond to one bits in d. The upper 12 bits of A are not altered.

```
A   001110101011001001
d               001010
    ------------------
    001110101011000011
```

LPN d — Logical product d — (12) — 12dd

Forms in the A register the bit-by-bit logical product of d and the lower 6 bits of A. The upper 12 bits of A are zero.

```
A   001110101011001001
d               001010
    ------------------
    000000000000001000
```

SCN d — Selective clear — (12) — 13dd

Clears the lower 6 bits of the A register where corresponding bits of d are ones. The upper 12 bits of A are not altered.

```
A   001110101011001001
d               001010
    ------------------
    001110101011000001
```

LMD d — Logical difference (d) — (12) — 33dd

Forms in the A register the bit-by-bit logical difference of the lower 12 bits of A and the contents of location d. Equivalent to complementing individual bits of A which correspond to one bits in the contents of d. The upper 6 bits of A are not altered.

```
A   001110101011001001
d         010100001010
    ------------------
    001110111111000011
```

LMI D	Logical difference ((d))	(12)	43dd

Forms in the A register the bit-by-bit logical difference of the lower 12 bits of A and the 12-bit operand obtained by indirect addressing. Equivalent to complementing individual bits of A which correspond to one bits in the operand. The upper 6 bits of A are not altered.

```
A      001110101011001001
((d))        010100001010
       ------------------
       001110111111000011
```

LPC c	Logical product c	(24)	22cc cccc

Forms in the A register the bit-by-bit logical product of the contents of A and the 18-bit quantity c.

```
A      001110101011001001
c      001110000011001010
       ------------------
       001110000011001000
```

LMC c	Logical difference c	(24)	23cc cccc

Forms in the A register the bit-by-bit logical difference of the contents of A and the 18-bit quantity c. Equivalent to complementing the individual bits in A which correspond to one bits in c.

```
A      001110101011001001
c      000010000000001010
       ------------------
       001100101011000011
```

LMM m,d	Logical difference M	(24)	53dd mmmm

Forms in the A register the bit-by-bit logical difference of the lower 12 bits of A and a 12-bit operand obtained by indexed direct addressing. Equivalent to complementing individual bits of A which correspond to one bits in the operand. The upper 6 bits of A are not altered.

```
A      001110101011001001
M            010100001010
       ------------------
       001110111111000011
```

REPLACE CODES — Place results of an arithmetic operation in the A register and destroy original contents of A register.

RAD d	Replace add (d)	(12)	35dd
	Adds the 12-bit quantity in location d to the contents of the A register and stores the lower 12 bits of the result back in location d. The result is left also in the A register at the end of the operation.		
AOD d	Replace add one (d)	(12)	36dd
	Adds one to the original value in location d and stores the lower 12 bits of the result back in location d. The result is left also in the A register at the end of the operation.		
SOD d	Replace subtract one (d)	(12)	37dd
	Subtracts one from the original value in location d and stores the lower 12 bits of the result back in location d. The result is left also in the A register at the end of the operation.		
RAI d	Replace add ((d))	(12)	45dd
	Adds A register contents to the operand from the location specified by the contents of d. The resultant sum is left in the A register at the end of the operation, and the lower 12 bits of A replace the original operand in memory.		
AOI d	Replace add one ((d))	(12)	46dd
	Adds one to the operand obtained from the location specified by the contents of d. The resultant sum is left in the A register at the end of the operation, and the lower 12 bits of A replace the original operand in memory.		
SOI d	Replace subtract one ((d))	(12)	47dd
	Subtracts one from the operand obtained from the location specified by the contents of d. The resultant difference is left in the A register at the end of the operation, and the lower 12 bits of A replaces the original operand in memory.		

RAM m, d	Replace add M	(24)	55dd mmmm
	Adds A register contents to the operand obtained from the location determined by indexed direct addressing. The resultant sum is left in the A register at the end of the operation, and the lower 12 bits of A replace the original operand in memory.		
AOM m, d	Replace add one M	(24)	56dd mmmm
	Adds one to the operand obtained from the location determined by indexed direct addressing. The sum is left in the A register at the end of the operation, and the lower 12 bits of A replace the original operand in memory.		
SOM m, d	Replace subtract one M	(24)	57dd mmmm
	Subtracts one from the operand obtained from the location determined by indexed direct addressing. The result is left in the A register at the end of the operation, and the lower 12 bits of A replace the original operand in memory.		

BRANCH CODES

The r subfield is a numeric value indicating the number of locations to a maximum of 31_{10} (37_8) to be jumped. If r is positive (01-37_8) the jump is forward r locations. If r is negative (40-76_8) the jump is backward r locations. If r equals 00 or 77_8, the program stops.

UJN r	Unconditional jump r locations	(12)	03rr
	Unconditional jump of up to 31 steps forward or backward from current program address, depending on value of r.		
ZJN r	Zero jump: jump r locations if (A) = 0	(12)	04rr
	Conditional jump of up to 31 steps forward or backward from current program address if A register is zero. If A is nonzero, the next instruction is executed. Negative zero (777777) is treated as nonzero.		

	NJN r	Nonzero jump: jump r locations if (A) $\neq$ 0 Conditional jump of up to 31 steps forward or backward from current program address if A register is nonzero. If A is zero, the next instruction is executed. Negative zero (777777) is treated as nonzero.	(12)	05rr
	PJN r	Plus jump: jump r locations if (A) $\geq$ +0 Conditional jump of up to 31 steps forward or backward from current program address if A register is positive. If A is negative, the next instruction is executed.	(12)	06rr
	MJN r	Minus jump: jump r locations if (A) $\leq$ -0 Conditional jump of up to 31 steps forward or backward from the current program address if A register is negative. If A is positive, the next instruction is executed.	(12)	07rr
	LJM m,d	Long jump to M Jumps to sequence beginning at address m + (d). If d = 0, m is not modified.	(24)	01dd mmmm
	RJM m,d	Return jump to M Stores the current program address plus two (P + 2) at location m + (d), and jumps to location m + (d) + 1.	(24)	02dd mmmm
CENTRAL PROCESSOR AND CENTRAL MEMORY CODES	EXN d	Exchange jump Transmits an 18-bit address from the A register to the central processor and directs the central processor to perform an exchange jump. The address in A is the starting location of a 16-word file containing information about the CP program to be executed. The 18-bit initial address must be entered in A before this instruction is executed. The	(12)	260d

EXN d (Cont'd)	central processor replaces the file with similar information from the interrupted CP program. The PP program is not interrupted.		
MXN d	Monitor exchange jump	(12)	261d
	Conditional exchange jump to the CP initiates CP monitor activity. If the monitor flag bit is clear, this instruction sets the flag and initiates the exchange. If the monitor flag bit is set, this instruction acts as a pass instruction. The starting address for this exchange is the 18-bit address in the PP A register which is an absolute address. The PP program must have loaded its A register with an appropriate address prior to executing this instruction.		
RPN	Read program address	(12)	2700
	Transfers contents of the central processor program address (P) register to the PP A register. Allows the PP to determine whether the central processor is running.		
CRD d	Central read from (A) to d	(12)	60dd
	Transfers a 60-bit central memory word to 5 consecutive PP memory locations. The A register must contain the 18-bit absolute CM address before the instruction is executed. The 60-bit CM word is disassembled into five 12-bit words beginning at the left. Location d receives the first 12-bit word. The remaining 12-bit words go to succeeding locations. The A register contents are unchanged.		
CRM m,d	Central read (d) words from (A) to M	(24)	61dd mmmm
	Reads a block of 60-bit words from CM into PP memory. The A register contains the 18-bit CM starting address and must be loaded prior to the execution of this instruction. The contents of A are increased by one as each 60-bit CM word is disassembled and stored. The		

CRM m,d (Cont'd)	block length or number of CM words to be read is contained in location d. The number also goes to the Q register, an indexing register, where it is reduced by one as each CM word is processed. Transfer is complete when Q = 0. The current contents of the (P) register are stored in PP location 0000, and the PP starting address m in the P register, which is increased by one as each 12-bit word is stored. Five words are required for each CM word read, since each CM word is disassembled into five successive PP words. The original contents of P are restored upon completion of the transfer.		
CWD d	Central write from d to (A)	(12)	62dd
	Assembles five successive 12-bit words into a 60-bit word and stores it in CM. The 18-bit CM address must be in the A register prior to the execution of the instruction and remains there unchanged. The first word to be read out of PP memory is contained in location d. It appears as the leftmost 12 bits of the 60-bit word. The remaining 12-bit groups are taken from successive addresses in PP memory.		
CWM m,d	Central write (d) words from M to (A)	(24)	63dd mmmm
	Assembles a block of 60-bit words and writes them in CM. The A register contains the beginning CM address and must be loaded prior to the execution of this instruction. The number in A is increased by one after each 60-bit word is assembled to provide the next CM address. The contents of location d specify the number of 60-bit words to write. The number also goes to the Q register where it is reduced by one as each CM word is assembled. Transfer is complete when Q = 0.		

CWM m,d (Cont'd)	The original contents of the P register are stored in PP location 0000. The address of the first word to be read from PP memory, m, goes to the P register which is increased by one as each 12-bit word is read to provide the next PP memory address. The original contents of the P register are restored at the completion of the transfer.		

INPUT/OUTPUT CODES

AJM m,d	Jump to m if channel d active	(24)	64dd mmmm
	Conditional jump to a new program sequence beginning at address m if the channel specified by d is active. If the channel is inactive, the current program sequence continues.		
IJM m,d	Jump to m if channel d inactive	(24)	65dd mmmm
	Conditional jump to a new program sequence beginning at address m if the channel specified by d is inactive. If the channel is active, the current program sequence continues.		
FJM m,d	Jump to m if channel d full	(24)	66dd mmmm
	Conditional jump to a new program sequence beginning at address m if the channel specified by d is full. If the channel is empty, the current program sequence continues.		
	An input channel is full when the input equipment has sent a word to the channel register and sets the full flag. The channel remains full until the PP accepts the word and clears the flag. An output channel is full when a PP sends a word to the channel register and sets the full flag. The channel is empty when the output equipment accepts the word and notifies the PP.		

EJM m,d	Jump to m if channel d empty	(24)	67dd mmmm
	Conditional jump to a new program sequence beginning at address m if the channel specified by d is empty. If the channel is full, the current program sequence continues.		
IAN d	Input to A from channel d	(12)	70dd
	Transfers a word from input channel d to the lower 12 bits of the A register. The upper 6 bits are cleared. If this instruction is executed when the channel is inactive, the PPs will become inoperative until deadstart.		
IAM m,d	Input (A) words to m from channel d	(24)	71dd mmmm
	Transfers a block of words from input channel d to PP memory beginning at a location specified by m. The A register contains the block length which is reduced by one as each word is read. The input operation is complete when A = 0. The current contents of the P register are stored in PP location 0000 and the starting address, m, in P. As each word is stored P is increased by one to give the next address. The original contents of the P register are restored at the end of the operation. If this instruction is executed when the data channel is inactive, no input operation is accomplished; the program continues at P+2.		
OAN d	Output from A on channel d	(12)	72dd
	Transfers a word from the lower 12 bits of the A register to output channel d. The A register remains unaltered. If this instruction is executed when the channel is inactive, the PPs will become inoperative until deadstart.		

OAM m,d	Output (A) words from m on channel d	(24)	73dd mmmm

Transfers a block of words on output channel d from PP memory beginning at the location specified by m. The number of words is specified by the contents of the A register, which is reduced by one as each word is transferred. The output operation is completed when A = 0.

The current contents of the P register, m, are stored in PP location 0000. P is increased by one as each word is read to give the next address. The original contents of the P register are restored at the end of the operation. If this instruction is executed when the data channel is inactive, no output operation is accomplished; the program continues at P+2.

ACN d	Activate channel d	(12)	74dd

Activates the channel specified by d. This instruction must precede instructions 70dd-73dd mmmm. Activating a channel alerts the input/output equipment for the exchange of data. Activating an already active channel causes the PP to become inoperative until deadstart.

DCN d	Disconnect channel d	(12)	75dd

Deactivates the channel specified by d. Stops the input/output equipment and terminates the buffer. Deactivating an already inactive channel causes the PP to become inoperative until deadstart. Care must be taken to avoid disconnecting the channel before first sensing for Channel Empty, deactivating a channel before stopping the associated processor, and deactivating a channel before putting a useful program in the associated processor. After deadstart, PPs wait on an input channel. Deactivating a channel after deadstart causes an exit to address 0001 and execution of program.

FAN d	Function (A) on channel d	(12)	76dd
	The external function code in the lower 12 bits of the A register is sent out on channel d. Executing this instruction when the channel is active causes the PP to become inoperative until deadstart.		
FNC m,d	Function m on channel d	(24)	77dd mmmm
	The external function code specified by m is sent out on channel d.		

PSEUDO INSTRUCTIONS

Pseudo instructions are grouped here according to general function. Their appearance in a subprogram is governed by the following rules:

1. Operations required:

   IDENT must be the first line

   END must be the last line

2. When the following operations are used, they must appear before any operations listed at 4. They must also appear before a macro call or the pseudo operation HERE if either generates an operation listed at 4.

   ABS

   PERIPH

3. The following operations may appear anywhere between IDENT and END:

   MACRO, its definition, and ENDM

   Comments lines

   LIST, EJECT, SPACE, TITLE, ERR, LCC, XTEXT, RMT and its bracketed code

   HERE and XTEXT, only if code generated does not include operations listed at 4.

   A macro call only if it does not expand into any of the operations listed at 4.

4. The first appearance of these operations makes illegal the subsequent appearance of ABS or PERIPH.

   USE, LOC, ORG
   MICRO
   Any machine instruction
   ENTRY, EXT
   EQU, SET
   BSS, BSSZ
   DATA, VFD, REP, DIS
   DUP, ENDD, STOPDUP
   All conditional pseudo instructions
   SST

5.1 ASSEMBLER CONTROL

The mode of assembly is controlled by these instructions.

5.1.1 IDENT

The first operation of every subprogram must be IDENT.

Location	Operation	Variable
ignored	IDENT	1, 2, or 3 subfields

IDENT can occur only once in each subprogram. Any additional occurrence is considered an error. If it is omitted, an error will result. The first variable subfield must contain a linkage symbol which becomes the name of the subprogram only and is not defined in the assembly (see section 3.3). For relocatable assemblies, the second and third subfields are ignored.

If the assembler is called by FORTRAN rather than a COMPASS control card, IDENT must appear in columns 11-15.

In absolute assemblies, the second subfield defines the first word address of the absolute binary program image. During assembly, data may be originated at a location higher than the base origin address, but not below it. This first word address does not serve the same function as an ORG nor does it replace ORG to set the origin counter value. A second subfield on the IDENT line is evaluated as a decimal number unless specifically designated octal.

In absolute CP subprograms, the third subfield contains the entry address. Assembler binary output is explained in section 8.

If the TITLE instruction is omitted, the IDENT variable field is used for the main subprogram title.

5.1.2 END

END is required as the last operation of each subprogram.

Location	Operation	Variable
symbol or blank	END	blank or a linkage symbol

This operation terminates a subprogram deck. It causes the assembler to terminate any counter, conditional assembly, macro generation or code duplication in progress. Any waiting remote text is assembled; all local blocks are assigned an origin relative to the program origin in the order in

which they were first introduced. If the location field of END contains a symbol it is defined as having a relocatable value equal to the total subprogram length, or at last word address + 1. Total subprogram length includes the length of the literals block. A symbol in the variable field of END is considered a transfer address and is relevant only for relocatable assemblies. This transfer address defines the starting point of execution of a program when it is loaded.

5.1.3 ABS

A non-relocatable CP program may be assembled with this instruction:

Location	Operation	Variable
ignored	ABS	ignored

ABS declares the program to be absolute; if used, it must appear at the beginning of the assembly. The assembler assigns all blocks an origin relative to absolute zero. Although the output is absolute, relocatable symbols may exist during assembly. Any literal or any symbol defined in a block other than the zero block is considered relocatable. This is a relevant consideration for symbol definition, storage allocation, and the IF pseudo instruction.

In absolute assemblies, ENTRY, REP, REPI and EXT are illegal.

5.1.4 PERIPH

PP code is assembled with this instruction:

Location	Operation	Variable
ignored	PERIPH	ignored

PERIPH declares the program to be a PP program and absolute. The rules stated under ABS apply; in addition, LCC is illegal.

Within PERIPH assemblies, the register names of CP assemblies are treated as normal symbols. Any CP instruction will cause an operand error.

5.1.5 BASE

With BASE, the programmer can change the mode of numeric data.

Location	Operation	Variable
ignored	BASE	O or D

A variable field symbol beginning with the letter O denotes octal assembly mode; a symbol beginning with D denotes decimal mode. Any other entry will be flagged as an error, and assembly will be decimal.

In succeeding lines, all numeric address constants and data items consisting of digits without O, D or B prefix or suffix are subject to the base mode control. Under BASE O, for example, the constant 15 is considered 15_8, as is 15B, and constant 15D is evaluated as 17_8. In octal assembly mode, any numeric item containing an 8 or 9 without a D prefix or suffix is flagged as an error. Decimal assembly mode is always assumed if no BASE is encountered.

All numeric items are under base control (except scale factors and binary point position which are always considered decimal items). For example, using the octal assembly mode, VFD 60/-1 defines a 48-bit field. A second subfield on the IDENT line is evaluated as a decimal number unless specifically designated octal.

5.1.6 SEGMENT

This pseudo instruction is used for producing central processor and peripheral processor overlays at assembly time. SEGMENT can be used only in a PP assembly or an absolute CP assembly.

Location	Operation	Variable
Segname	SEGMENT	orgbase, eptname

Segname is the name of the overlay and must be present for the loader as a linkage symbol. Segname is not defined in the assembly.

Orgbase defines the first word address of the absolute binary program image. During assembly, data may be originated at a location higher than the base origin address, but not below it. This first word address does not serve the same function as an ORG nor does it replace ORG to set the origin counter value.

Eptname indicates the entry address to the segment. The SEGMENT pseudo instruction causes COMPASS to write, to the binary output file, all binary information accumulated since the previous IDENT or SEGMENT card was encountered and to write an end-of-record. The binary information consists of blocks, literals, and assembled code. The symbol table is not cleared after encountering SEGMENT.

SEGMENT should be used in conjunction with a USE or ORG pseudo operation to indicate the location where the segment is to be loaded.

In a CP assembly, a BSS 1 is required as the first instruction in the segment (after ORG and USE) to allow room for a control word to be loaded into the first word prior to the orgbase. (appendix F)

Examples :

1.

```
OVLOC     BSS       0                  Location where
          .                            segment is loaded
          .
          .
SEG1      SEGMENT   STRTLOC,ENTPNT
          ORG       OVLOC
          BSS       1                  First address of
STRTLOC   BSS       0                  segment binary
          .                            information
          .
          .
                (tables)
ENTPNT    BSS       0                  Entry point of
          .                            segment
          .     (program)
          .
```

The segment, SEG1, will be loaded as an overlay. The first word address of the binary information to be loaded is STRTLOC. The entry point to the overlay and the first executable instruction is location ENTPNT. The overlay, when executed, will occupy the area beginning at location OVLOC.

2.

```
SEGA      SEGMENT   STRTLOC,ENTPNT
          USE       BLOCK1             assemble in block
          BSS       1                  1 used by loader
STRTLOC   BSS       0
          .
          .                            (tables)
          .
ENT       BSS       0
          .
          .                            (program)
          .
```

The segment, SEGA, will be loaded as an overlay. The first word address of the binary information to be loaded is STRTLOC. The entry point and first executable instruction is ENTPNT. The overlay will be assembled in the block, BLOCK1, and when executed will occupy an area relative to the block origin.

All segment overlays are level (1,0). If errors occur or if word count is zero, no binary data will be dumped.

The programmer must set up the necessary loader call, overlay level, and the type of load requested. (See SCOPE 3.1 Reference Manual.)

5.2 COUNTER CONTROL

These pseudo instructions control the origin, location and position counters.

5.2.1 USE

USE declares a block into which succeeding instructions are to be placed.

Location	Operation	Variable
ignored	USE	block name

Upon encountering USE, the assembler places succeeding assembled values in the block named in the variable field. The first appearance of a block name in USE causes a force upper, subsequent USE statements for that block do not. The values of the current origin and position counters are saved to indicate the last known length of the block being assembled. An indication as to whether the next instruction is to be forced upper is also saved. If the block name in the USE statement is enclosed in slashes, that block is a common block, and subsequent uses of that name in USE need not be enclosed in slashes. If the block name is never enclosed in slashes, it is a local block.

The following notations may be used to set the origin of data:

USE	Data origin is in zero block
USE 0	Data origin is in zero block
USE //	Use blank common block
USE *	Use block in effect prior to current USE

A common block can be declared with the same name as a local block. In such cases, the common block name must be enclosed in slashes in subsequent USE statements to distinguish it from the local block. Thus, common block zero can coexist with the program's zero block if it is referenced always in the following manner:

```
USE / 0 /
```

The zero block, the nominal program block, contains the entire program if no other USE is encountered.

If the blank common block is named in a USE statement, BSS and ORG are the only storage allocation instructions that may follow USE; BSSZ is not permitted since it presets the block to zero.

The assembler maintains a record of USE and ORG pseudo operations since each occurrence of these pseudo operations (except USE*) adds an entry to this record. Each use of USE * restores the most recent entry and removes it from the list. In this way, a push-down list is maintained. Only the last 50 entries are maintained. When the list is exhausted (more USE * instructions than entries), the zero block is used.

Any symbol used as a block name has definition as a block name only, and may be defined elsewhere without ambiguity.

If a USE statement introduces a block name that has not appeared previously in a USE statement, the origin and location counters are started at zero relative to the block origin, and the position counter is set to the beginning of a new word. Block type is considered local unless the block name is enclosed in slashes.

If the block name has previously appeared in a USE operation, or is the zero block, the origin, location, and position counters are started at their last known values.

If the last instruction assembled under this block was one which forces the next instruction upper, that last instruction will be forced upper. For example:

```
GAMMA   RJ      ALPHA
        USE     DATA
SAN     DATA    1.0
        USE     *
        SA3     SAM
```

The SA3 instruction will be forced upper.

If the last instruction did not indicate a force upper, forcing upper is determined by the instructions which follow USE. With this facility, partial-word bytes may be packed into a table which resides in a block other than the one currently being used. For example:

```
.
.
.
USE    /TABLE/
VFD    6/CODE
USE    *
.
.
.
USE    /TABLE/
VFD    6/1RX,18/ADDR
USE    *
.
.
.
```

The value of the location counter is not saved; if LOC has been employed, caution must be exercised to produce the desired results.

When assembly takes place within a block, that block name in a USE statement has the effect of forcing the location counter to agree with the origin counter and recording this block as the last known block for a subsequent USE *.

5.2.2 ORG

With ORG, the origin and location counters may be reset.

Location	Operation	Variable
ignored	ORG	address expression

The ORG instruction causes the location and origin counters to be reset to the value stated in the address field. As in USE, the current origin, location, and position counters are saved. ORG starts the assembly at the upper position of a word.

The only effect of * in the variable field of ORG is to force the current block upper. The USE pseudo instruction must return control to the last used block.

The expression in the variable field of ORG must not contain symbols not yet defined; the expression may not result in a negative relocatable value.

5.2.3 LOC

The location counter may be set with this instruction.

Location	Operation	Variable
ignored	LOC	address expression

The location counter is set to the value of the variable field expression, but the origin counter is not reset. Normally the location counter value is the same as the origin counter, since instructions are executed normally at the location into which they were loaded. LOC allows the location counter to be adjusted so that code may be loaded into one place, and executed at another. The location counter is reset to origin counter value when a subsequent USE or ORG is encountered.

Symbols in the variable field expression of LOC must have been previously defined. LOC causes the next instruction to be forced upper. The only effect of LOC * is to force upper.

5.3 LINKAGE CONTROL

Names to be passed to the loader for subprogram linkage are declared with these instructions. They are valid for relocatable code only, and may not exceed seven characters in length.

5.3.1 ENTRY

An entry point name is passed to the loader with this statement.

Location	Operation	Variable
ignored	ENTRY	symbols separated by commas

The linkage symbols listed in the variable field are declared to the loader as entry points. Each must be defined in the assembly as a non-external symbol.

5.3.2 EXT

This instruction declares symbols external to the subprogram.

Location	Operation	Variable
ignored	EXT	symbols separated by commas

The linkage symbols listed in the variable field are passed to the loader as external symbols. These symbols must not be defined within the subprogram.

5.4 STORAGE ALLOCATION

These pseudo instructions cause adjustment of both the location and origin counters. All operations force upper.

5.4.1 BSS

A storage area is reserved with this statement:

Location	Operation	Variable
symbol or blank	BSS	absolute address expression

A location field symbol is defined as the current value of the location counter. The expression in the address field is evaluated and the location and origin counters are incremented by that amount. Symbols in the expression must have been previously defined. If the address expression is incorrect, no space will be reserved, but a force upper will occur. BSS 0 forces upper without allocating storage.

5.4.2 BSSZ

BSSZ reserves an area of zero-filled words in storage. The specification of BSSZ is similar to BSS, and the effect is the same, except that allocated storage is preset to zeros at load time. BSSZ 0 forces upper without allocating storage.

5.5 SYMBOL DEFINITION

These operations permit the direct definition of symbols.

5.5.1 EQU

Location	Operation	Variable
symbol	EQU	address expression

The symbol in the location field is defined as having the same value as the address expression. Once defined, the symbol retains that definition throughout assembly. An undefined symbol may not appear in the variable field expression. (=Ssymbol and =Xsymbol may not be used in the address field unless the symbols have been defined by some other conventional method.) The address expression may result in an absolute, relocatable, or external value. If the address field is incorrect, the location symbol of the EQU is not defined, and a warning flag is issued.

5.5.2 SET

Location	Operation	Variable
symbol	SET	address expression

SET redefines the value of the location symbol to the value of the variable field expression. Such symbols are called redefinable and may be defined only with the SET instruction; they have this definition only until reset. Symbols in the address expression must have been previously defined. A SET-defined symbol may not be referenced before it is first defined by a SET. (=Ssymbol and =Xsymbol may not be used in the address field unless the symbols have been defined by some other conventional method.) The address expression may result in an absolute, relocatable or external value. If the address field is incorrect, the location symbol is not redefined, and a warning flag is given.

5.6 DATA GENERATION

With these instructions, data items may be included in the subprogram.

5.6.1 DATA

The DATA operation declares numeric and character data items.

Location	Operation	Variable
blank or symbol	DATA	absolute data items

If a location symbol is present, it is defined as the current value of the location counter. The data items may be octal, decimal, or display code characters, and must be full-word values. They are separated by commas and terminated by a blank. Literals may not be used in the variable field list. The DATA pseudo instruction forces upper. Refer to section 3 for specification of data items.

5.6.2 DIS

DIS provides a convenient means of writing display code lines when more than one COMPASS statement is involved.

Location	Operation	Variable
blank or symbol	DIS	word count, and a character string

The word count must result in an absolute value. COMPASS extracts n·10 characters beyond the comma following the address expression, and packs them, as they occur, into n words. If the statement ends before n·10 is satisfied, the remainder of the words requested will be filled with blanks (55_8). (For PP, n·2 is the character count.)

If the count subfield is missing or has a zero value, the character string must be bounded by delimiters. The comma must always be present. The first character after the comma is the delimiter. All characters between the delimiter and its next occurrence are packed into as many words as are necessary. Two zeros are guaranteed at the end of the character string; COMPASS allocates another word to accommodate them if required. If the delimiter character is not encountered again, COMPASS will produce a fatal error.

The DIS pseudo instruction forces upper.

5.6.3 LIT

Absolute values are entered into the literal table with the LIT statement.

Location	Operation	Variable
blank or symbol	LIT	up to 100 words of data items

A location symbol indicates the location of the first mentioned value. Data items are separated by commas and terminated by a blank. Data items are entered in the literal table in the order specified. Duplications in data items may occur in the literal table if there are duplicate values in the LIT variable field which occur in a different sequence; but if <u>all</u> data items listed for one LIT are identical to an existing sequence in the literal table, they will not be duplicated. Subsequently defined literals (defined either with LIT or the =n form) will not be duplicated in the literal table if they exist in a LIT declared sequence.

The specification of data items in the LIT variable field is the same as for DATA. No = is used before LIT-declared literals. At least one data item must be specified.

5.6.4 VFD

Fields of binary data are generated with the VFD statement.

Location	Operation	Variable
blank, +, -, or symbol	VFD	a list of subfields separated by commas

When plus or a symbol appears in the location field, data begins in a new word. A symbol is given the new value of the location counter. A minus sign in the location field causes the position counter to be set at the next quarter word boundary in a CP assembly, or at a new word in a PP assembly.

The subfields are of the format n/v where n is a bit count of field length and may be any single, previously defined, absolute element. It must be positive and may not exceed 60. The value expression, v, consists of any valid address expression. If a non-absolute value (v) occurs (relocatable or external), it must be within a field that is at least 18 bits long and ends at bit 0, 15, or 30.

Absolute data items follow all rules indicated in section 3.7 and are right or left justified within the field length.

Example:

```
ALPHA    SET    15
TABLE    VFD    36/4CTAB1,6/9,18/TABLOC
         VFD    30/*-1,30/5HΔΔΔΔΔ,ALPHA/-0
         VFD    $/0,1/1
```

Word 1	2 4 0 1 0 2 3 4 0 0 0 0	1 1	T A B L O C
Word 2	0 0 0 0 T A B L E	5 5 5 5 5 5 5 5 5 5	
Word 3	7 7 7 7 7	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	

VFD leaves the position counter pointed at the next available bit position. If the last VFD byte ends to the left of a quarter word boundary, zero bits will be inserted up to a quarter word boundary. If one VFD instruction is followed immediately by another which has no location field entry or if the two VFDs are separated only by a USE...USE * routine, values are packed into words with no padding or forcing upper.

A plus or minus in the location field of a VFD in PP forces the VFD data to begin at the next full word boundary.

5.6.5 REP

REP defers data generation until load time. It is valid only in relocatable assemblies.

Location	Operation	Variable
ignored	REP	1 to 5 subfields separated by commas

Information is passed by the assembler to the loader. This replication control is used when a block of storage is to be set to a given series of values, yet is not to be represented in its duplicated state in the COMPASS binary output. A BSSZ instruction with an address area greater than five is output in a REPI table. First a set of data is placed in consecutive locations, established by the programmer using normal assembler techniques. Then the loader is instructed to move blocks of data in storage. For this, five values are specified in the REP or REPI instruction. For REPI the non-relocatable data must appear in previously loaded text. This data must not contain any external references or common relocatable addresses. Each subfield consists of a letter, S, D, C, B, I, and a slash, followed by a non-external address expression.

S Source address

D Destination address

C Repetition count

B Code block size

I Increment

The operation at load time is to move B words from location S to location D, B words from location S to location D + I, B words from S to location D + 2I, etc. This operation is repeated C times. An omitted specification, except S, is passed to the loader as zero. Only one specification of each type may appear. If a subfield is zero, the loader will make the following assumptions, in the order shown:

B = 1

I = B

C = 1

D = value of S subfield plus value of B subfield

If the value of S is zero, the assembler will flag the REP or REPI instruction as an error and will not pass REP or REPI to loader.

The loader tables produced by the REP and REPI instructions differ only in one byte. The REPI table is processed by the loader upon encounter, whereas the processing of the REP table is delayed until the closing out of load.

The assembler error-flags the REP instruction if the value of S is zero, and does not pass REP to the loader.

At load time, REPs are deferred until all other loading is finished.

5.7 CONDITIONAL OPERATIONS

These pseudo instructions control the conditional assembly of code; succeeding instructions are assembled only if the condition stated is true. When a value of an address expression is involved, only previously defined symbols may be used, and the result must not be relocatable. If undefined elements are used, the expression has a zero value, the conditional is flagged as an error and assembly proceeds with the next instruction.

The number of instructions to be assembled or skipped may be controlled by a line count or by brackets (an IF to a matching ENDIF). A count of the number of statements assembled under control of an IF statement can be included as the last subfield. If the count field is missing or zero, the assembler looks for a bracketing ENDIF, and assembly resumes with the instruction immediatly following it. Comment lines with an asterisk in column 1 are not included in the count. The skip count is decremented only for instruction lines. Comments which occur before the first instruction following the skipped instruction are skipped also.

If there is an instruction bracket name, the corresponding ENDIF is the first one encountered which has either the same name as the IF or no name. If there is no instruction bracket name on the conditional instruction and no line count is given, the first ENDIF encountered, with or without a name, terminates the bracket. Instruction brackets have significance only if coding is <u>not</u> to bc assembled. An ENDIF encountered during assembly is ignored. An END card terminates the skipping process. During the skipping process, macros are not expanded: an ENDIF which would have had effect in the macro expansion is ignored.

Conditional pseudo instructions can:

Test comparative value of two address expressions

Test assembly environment

Test the attribute of a single symbol or address expression

Test the value of character strings

5.7.1 IF: COMPARE EXPRESSION VALUES

Location	Operation	Variable
blank or instruction bracket name	IFxx	2 or 3 address expressions separated by commas

xx is EQ, NE, GT, GE, LT, or LE. The values of the first two address expressions are compared. The third is the number of lines to be assembled if the comparison is satisfied.

IFEQ: Succeeding code is assembled if the values are equal.

IFNE: Succeeding code is assembled if the values are not equal.

In IFEQ and IFNE tests, all information pertinent to the value of the two address expressions is compared for equality. Not only must the expressions have the same numeric value, but they must have equal attributes. For example, both must be common relocatable, program relocatable, absolute, external, or register names.

IFGT: Succeeding code is assembled if the value of the first subfield is greater than the second.

IFGE: Succeeding code is assembled if the value of the first subfield is greater than or equal to the second.

IFLT: Succeeding code is assembled if the value of the first subfield is less than the second.

IFLE: Succeeding code is assembled if the value of the first subfield is less than or equal to the second.

In the last four tests, only the values of the expressions are compared. Relocation and other attributes are not tested for equality.

5.7.2
IF: TEST ASSEMBLY ENVIRONMENT

Location	Operation	Variable
blank or instruction bracket name	IFPP or IFCP	A single optional address expression

IFPP tests for a PP assembly; IFCP tests for a CP assembly. The variable field expression results in a count of lines to be skipped if the test is not satisfied.

5.7.3
IF: TEST SYMBOL ATTRIBUTE

Location	Operation	Variable
blank or instruction bracket name	IF	2 or 3 subfields, separated by commas: attribute mnemonic, symbol or address expression; address expression

The attribute mnemonic is SET, ABS, REL, REG, EXT, COM, LOC or DEF. The symbol or address expression depends on the mnemonic used. The address expression results in the line count. The line count and its preceding comma may be omitted if ENDIF is used.

Negative attribute may be specified by preceding the attribute mnemonic with a minus sign.

The following tests are made:

SET	Satisfied (true) if the symbol in the second subfield has been previously defined by the SET pseudo instruction; -SET is satisfied if the symbol is defined by any other method. The second subfield must be a single symbol.
ABS	Satisfied if the address expression is absolute (not relocatable or external). -ABS is satisfied if the expression value is not absolute.
REL	Satisfied if the address expression is common or program relocatable. -REL is satisfied if the address expression is other than program or common relocatable.
REG	Satisfied if any symbol in the address expression is a register name. -REG is satisfied if no symbol is a register name.
COM	True if the expression is common relocatable. -COM is true if the expression is not common relocatable.
EXT	True if any symbol in the address expression is an external symbol. -EXT is true if there is no external symbol.
LOC	Satisfied if the expression is program relocatable. -LOC is true if the expression is not program relocatable.
DEF	Satisfied if all symbols in the expression have been defined. -DEF is satisfied if any expression symbol has not yet been defined.

The attributes listed, except -DEF and REG are known to the assembler only after the symbols in the expression have been defined. For example, if a common block name has not yet been declared in a USE pseudo instruction, a test for COM on that name will fail. Any test on an undefined symbol, except for DEF, REG or EXT, results in an error.

5.7.4 IFC

This option tests the equality of two character strings.

Location	Operation	Variable
blank or instruction bracket name	IFC	2 or 3 subfields separated by commas; (if no third subfield, second comma is omitted): a relational mnemonic, 2 delimited character strings, and an optional address expression

Relational mnemonics:

EQ or -NE	equal
NE or -EQ	not equal
GT or -LE	greater than
GE or -LT	greater than or equal
LT or -GE	less than
LE or -GT	less than or equal

The delimited character strings are of the format:

dccc...ccdccc...ccd

d is any character. Characters between the first and second d constitute the first character string; characters between the second and third d constitute the second character string.

The optional third subfield is an address expression which results in line count. It must be preceded by a comma. If ENDIF is used, the line count and its preceding comma may be omitted.

Each character in the first string is compared with the corresponding character in the second string, progressing from left to right, until an inequality is found or both strings are exhausted. If one string is shorter than the other, the short string is padded with a character which is smaller than any other character in the string.

The truth condition is evaluated on the relative magnitudes of the strings.

Example:

\$ABC\$ABC\$	is equal	\$A\$\$	is greater than
\$AB\$ABC\$	is less than	\$Z\$8\$	is less than
\$\$\$	is equal		

The collating sequence is given in Appendix A.

When IFC is used within a macro definition, one or both of the character strings may be a formal parameter name. For example, an IFC to check for an empty parameter string:

```
XX      MACRO     P1,P2
        IFC       EQ,**P2*,1
        ERR
         .
         .
         .
        ENDM
```

Since the character * is recognized as a formal parameter name delimiter, the catenation character → (sections 3.1.2 and 6.1.2) is not necessary. It would be required if the IFC delimiter character were not one of the characters + - * / $. ,) (= Δ. For example:

```
IFC     EQ,X→P1XX
```

5.7.5 IF: PP USAGE

The following example demonstrates a use of IF statements in a PP program:

```
        IF        DEF,LOOP,3
        IFLT      *-LOOP,40B,1
        ZJN       LOOP
IF2     IFGE      *-LOOP,40B
        NJN       *+3
        LJM       LOOP
IF2     ENDIF
```

This code assembles a zero jump to the symbol LOOP if LOOP has been defined within 31 (37 octal)† words prior to the occurrence of this code. The first conditional causes the next three statements to be assembled only if LOOP has been defined. If LOOP has not been defined, the other two conditional statements and the zero jump are skipped and a nonzero long jump is assembled. IFLT and IFGE are mutually exclusive; the code following only one of them can be assembled.

---

†The range of a short jump.

The IFGE conditional uses an ENDIF to bracket the code to be omitted if the test is not satisfied. The bracket name IF2 associates the ENDIF with the conditional. If, as in this example, other conditional coding is not overlapping, the bracket name is not required.

5.7.6 ENDIF

ENDIF terminates the range of a conditional assembly operation.

Location	Operation	Variable
blank or instruction bracket name	ENDIF	ignored

ENDIF terminates an instruction bracket; if it does not follow an instruction bracket, it is ignored. An ENDIF with no name terminates any conditional in effect. A named ENDIF terminates a conditional with the same bracket name, or a conditional with no name. ENDIF is ignored if it appears within a range controlled by line count.

5.8 LIST CONTROL

These instructions control the listing format. The listable output from a COMPASS assembly normally contains the following:

Heading information	Program length, origin and length of each block, entry points, external symbols.
Assembly text	Line and assembly results of each line assembled (not skipped) from the input device (not generated by RMT, DUP, XTEXT, or a macro expansion). For generative pseudo instructions (DATA, DIS, VFD), only one line is listed. Any line with an error flag is listed. Each line with the instruction LIST is listed.
Assembler statistics	Size of unused storage, a count of statements generated in assembly; if nonzero, a count of references discarded because of restricted core storage.

Error directory	Explanation of each error as well as the page on which it occurred. If no errors occur, the error directory is suppressed.
Reference table	List of each symbol, its definition, and for each reference, the value of the origin counter at the place of reference.

Primary list control is specified on the COMPASS control card. When L=0, only the heading information, assembler statistics, error-flagged lines and the error directory are listed. When L is other than 0, more extensive listings may be specified with the LIST pseudo instruction.

5.8.1 LIST

This instruction controls the listable output from COMPASS, and is relevant only if listings are being produced.

Location	Operation	Variable
ignored	LIST	list control options, separated by commas

Each option is represented by a single letter. Specifying the letter selects the option; the option may be discontinued by specifying the letter preceded by a minus sign. Normally the L and R options are on, all other options are off.

L List Control

Master list control. When not selected, only error-flagged lines and the LIST pseudo instruction are listed. The accumulation and listing of the reference table is not affected by this option.

R Reference Accumulation and List

When this option is not selected, no references are accumulated. If a complete reference listing is to be obtained, R should never be turned off. If off at the end of assembly, the reference table listing is suppressed.

G Code Generation List

When this option is selected, code generating lines are listed regardless of other list controls (except L). In this way, the code generated from macro calls may be listed without listing the entire macro expansion. Operations controlled by G include: machine operations, DATA, BSS, BSSZ, VFD, DIS.

A Assembly List

Normally (A not selected), when a → or ≠ mark appears in a line that would be listed, the line appears with the → and ≠ marks in it exactly as presented to the assembler. When the A option is selected, the catenation marks are removed and micros substituted.

N Symbol List

If selected, non-referenced programmer-defined symbols are listed.

T SST Symbol List

If selected, the non-referenced system symbols (SST) are listed.

C Control Card List

EJECT, SPACE, and TITLE are listed when this option is selected.

D Detail

The following items are listed when this option is selected: second and subsequent lines of VFD, DATA, DIS; code is assembled remotely when HERE or END causes its assembly; a list of literals and deferred symbols at the end of the assembly.

E Echoed Lines

When this option is selected, all iterations of duplicated code are listed.

F IF-skipped Lines

This option generates the lines skipped by IF-type instructions.

M Macro

When selected, this option lists the lines generated by macro calls. This does not include system macro list control.

S Systems Macros

When the S option is selected, lines generated by systems macros are listed.

X XTEXT Lines

When selected, the X option lists lines generated as a result of an XTEXT pseudo instruction.

The list options A, C, D, E, F, M, N, S, T, and X cause a line to be listed only if all the options which apply to it are on. For example, if a DUP appears within a macro, its expansion will be listed only if both M and E are on. If a systems macro call is made within XTEXT text, its expansion will be listed only if X and S are both on. If the marks → or ≠ appear in external text inside a DUP bracket, the lines will be listed with → and ≠ removed only if A and X and E are all on.

5.8.2 EJECT

EJECT is an operation field entry; location and variable fields are ignored. EJECT advances paper before printing; page headings are printed and listing continues.

5.8.3 SPACE

Location	Operation	Variable
ignored	SPACE	address expression

The address field expression indicates line spacing for the listing. If the listing exceeds the number of lines on the page, an eject occurs, and listing resumes after the titles are printed on the next page.

5.8.4 TITLE

With this instruction the programmer establishes titles for listings.

Location	Operation	Variable
ignored	TITLE	character string

The character string starts at the column immediately following a blank after the E of the operation code and continues for 79 columns, or to end of the statement. The title is filled with blanks if less than 79 columns of text are provided. Beyond 79 columns, text is lost. The first TITLE instruction in a subprogram defines the primary title which appears on every page. Subsequent TITLE instructions generate subtitles. Except for the first TITLE, this instruction causes a page eject. A card containing only the word TITLE results in untitled listings. If TITLE is not specified, the variable field of the IDENT line is used as the main title.

5.9 CODE DUPLICATION

5.9.1 DUP

A sequence of lines may be replicated with this instruction.

Location	Operation	Variable
blank or instruction bracket name	DUP	1 or 2 address expressions separated by a comma

The first address expression specifies how many times a series of lines following DUP is to be assembled. Each assembly is identical to the first one. The lines to be assembled may be indicated in one of two ways: by an instruction bracket (DUP to an ENDD), or by a line count on the DUP instruction, which is the second address expression.

Code is skipped, not assembled, if the iteration count is zero.

Any legal operation is permissible within the range of DUP, except END. A comment card with a column 1 * will not be counted in the line count, if one is given and will not be duplicated.

Indefinite duplication of code is specified by an unobtainable iteration count and the STOPDUP statement. ENDD or line count is still necessary.

5.9.2 ENDD

ENDD terminates the range of a DUP if a line count is zero or not used.

Location	Operation	Variable
blank or instruction bracket name	ENDD	ignored

ENDD should follow the last line to be duplicated or skipped as specified in the DUP statement. An ENDD with no location field entry terminates any DUP in effect, including any inner DUP. An ENDD with an instruction bracket name terminates a DUP with the samc name or a DUP with no name, and every inner DUP.

ENDD is ignored if it appears anywhere except as a DUP terminator.

5.9.3 STOPDUP

STOPDUP may be used to stop the duplication process. Normally, it is used after a conditional operation which, when satisfied, indicates that no more duplications are needed.

Location	Operation	Variable
ignored	STOPDUP	ignored

When STOPDUP is encountered, duplication stops with the current iteration regardless of the iteration count. Once STOPDUP is encountered, code is assembled to the proper ENDD or to the end of line count.

STOPDUP is ignored outside a DUP range.

5.10 REMOTE ASSEMBLY

RMT generates symbolic instructions for assembly at a later time or place; it supplements the USE facility. Code following USE is assembled when it is encountered; code following RMT is assembled later at a point specified by the programmer. COMPASS stores the code, unassembled, until it is called. Symbols, macro definitions, micros, and block names defined within a remote section do not become defined until the remote section is assembled.

5.10.1 RMT

RMT introduces the section of symbolic instructions to be saved for later assembly.

Location	Operation	Variable
ignored	RMT	ignored

All instructions between the first and second RMT statements are saved for later assembly. Any instructions, except RMT, may be contained within RMT sections as long as their usc is legal when the remote lines are assembled. COMPASS takes no note of remote code at the time it is saved, except to recognize a second RMT instruction, which acts as an off switch. Alternate appearances of RMT act as on/off switches. However, within remote sequences, macro calls, catenation or micro substitution may specify RMT sequences, since expansion and substitution occur at assembly time and not at remote definition time.

5.10.2 HERE

When HERE is encountered, all saved remote code is assembled. HERE also clears the remote retention table so that the code is not called again. The instruction consists simply of the operation field entry HERE. Other fields are ignored. If, in the assembly of remote sequences, RMT pairs occur, the bracketed lines will be saved for later assembly when another HERE or END is encountered.

In the absence of USE within the remote sequence, the remote code is assembled under whatever block is in effect at the time HERE is encountered.

If HERE does not occur in a subprogram, any waiting remote lines are assembled when END is encountered but before END is processed. Any remote lines which might have been saved as a result of this last remote assembly will be lost.

5.11 LOADER CONTROL: LCC

Loader directives may be included only in a relocatable source program. They are passed along in the binary output file for subsequent loader recognition. Loader directives are specified by LCC.

Location	Operation	Variable
ignored	LCC	any string of non-blank characters

All characters in the variable field from the first non-blank to the first blank are considered the directive. They are moved to the first position (column 1) of a loader table in packed display code. COMPASS does not edit the directive. Illegal forms are recognized at load time by the loader.

All loader directives appear before any of the binary output for a subprogram. For loader directive formats, refer to SCOPE documents.

5.12 ERR

ERR introduces a fatal error into the subprogram to inhibit subsequent loading.

Location	Operation	Variable
ignored	ERR	ignored

The appearance of ERR in a subprogram does not affect other code. It may be used in conjunction with a conditional assembly pseudo operation to force an error into the assembly based on a time test. This combination can be used effectively to check for illegal macro parameters.

5.13 EXTERNAL TEXT

XTEXT provides a method of introducing records from a file other than that being used for input.

Location	Operation	Variable
file name	XTEXT	blank or a record name

COMPASS gains access to the file named in the location field and searches for the named record. The contents of that record to an END card or end-of-record, are brought into the subprogram for assembly at the point where XTEXT is encountered. The text may contain any legal library macros for assembly, including macro definitions.

If the record name is not specified, COMPASS rewinds the file and reads only the first record in the file. If the record name is given, the file must be an indexed file with named records. If the file or the record cannot be found, an error flag is issued. The file must be a standard coded file exactly like an input file. Text brought in by XTEXT is not listed (except for lines with assembly errors) unless the X list option is selected.

5.14 SYSTEM SYMBOLS

SST permits definition of system symbols from the system file in the routine.

Location	Operation	Variable
ignored	SST	ignored

The system symbols define system functions such as system table pointers, PP resident entry pointers, monitor functions and direct PP locations. These symbols are used in system communication between the PPs and central memory resident.

The symbols exist on a system text file. The file is accessed through the S option on the COMPASS control card.

MACROS

A macro is a sequence of code that may be called whenever needed by a single instruction — a macro name. A macro name in the operation field of a statement (a macro call) results in the macro code sequence being assembled at that point in the program. The macro call may also contain parameters which are substituted for defined parameters in the macro code sequence. The use of a macro requires two steps: defining the macro sequence and calling the macro.

6.1 MACRO DEFINITION

A macro definition consists of three parts:

Macro heading	MACRO pseudo instruction which states the name of the macro and identifies its substitutable parameters. The LOCAL pseudo instruction may also be used to identify local parameters.
Macro body	Symbolic instructions which constitute the macro code sequence.
Macro terminator	ENDM pseudo instruction which terminates the definition.

A macro definition may appear anywhere in a subprogram before the macro is called. The definition is governed by the rules for pseudo instructions given in section 5.

A macro may be redefined at any time, the latest definition of a macro name applies to a macro call. For any redefinition, including redefining a mnemonic, a flag is issued but the new definition is valid.

6.1.1 MACRO HEADING

The macro heading line has two forms.

Standard Form

Location	Operation	Variable
macro name	MACRO	up to 63 parameters

The location field contains the macro name which may be any legal name except END, LOCAL, or ENDM; it may be the same as other program-defined symbols since it has meaning only in the operation field. For example, ABC may be a symbol as well as a macro name.

If a macro name is identical to a machine or pseudo instruction mnemonic, the mnemonic is redefined as the macro. For example, definition of a macro name SB3 overrides the machine mnemonic SB3; an SB3 in the operation field of a subsequent statement is interpreted as a macro call. If SB3 appears in the macro body it also is interpreted as a macro call and an infinite macro expansion may occur. Once a mnemonic has been redefined as a macro, there is no way of returning that name to mnemonic status. The macro may be redefined, however, to produce equivalent results by using a VFD.

The variable field of the MACRO line contains the name of substitutable parameters in the order in which they occur on the macro call instruction. Each is a symbol of one to eight alphanumeric characters beginning with a letter. Parameters are separated by any one of the following special characters; and the list is terminated by a blank. These special characters have no meaning other than as separators.

```
. , + - * / ) ( $ =
```

ENDM, LOCAL, or END may not be used as parameter names. Parameter names may occur more than once in the parameter list but subsequent appearances are ignored. Parameter names beginning with a number are ignored. The total number of unique parameter names plus LOCAL symbols may not exceed 63 for any one macro definition.

The following notations are all equivalent:

```
SUM    MACRO    X=Y+Z+X
SUM    MACRO    X(Y+Z)
SUM    MACRO    X=Y+Z
SUM    MACRO    X,Y,(Z+X)
```

The following are equivalent also:

```
RAO    MACRO    X
RAO    MACRO    X=X+1
```

Alternate Form

Location	Operation	Variable
blank	MACRO	2 or more subfields

This form is identified by the blank location field of the MACRO line. The macro name is the first subfield of the variable field. Subsequent subfields are the substitutable parameters, listed with the rules that govern the normal MACRO header form. The first of these substitutable parameters must be present in the alternate form macro. It is called the location argument since the location field entry of the macro call is its substituted value.

Example:

```
        MACRO   TABLE,TABNAM,VALUE1,VALUE2,
TABNAM  VFD     60/VALUE1,60/VALUE2
        ENDM
```

The macro is named TABLE, its substitutable parameters are TABNAM, VALUE1, and VALUE2. TABNAM is the location argument. TABLE might be called with an instruction like this:

```
SPVAL   TABLE   1.0,2.0
```

which will result in the expansion

```
SPVAL   VFD     60/1.0,60/2.0
```

If it had been called with this instruction:

```
        TABLE   1.0
```

the expansion would be

```
        VFD     60/1.0,60/
```

since the location argument and VALUE2 are null.

If the location argument is not present on the MACRO line, a warning flag will be given and the definition ignored. Therefore, the following examples of definition headers are illegal:

```
        MACRO   ABC
        MACRO   ABC,,FP
```

One or more LOCAL pseudo instructions may immediately follow the MACRO line of either form.

Location	Operation	Variable
ignored	LOCAL	list of symbols

The listed symbols may be separated by any one of the special characters:

, + - * / .) ($ =

Therefore a local symbol may not contain any of those characters.

The symbols are to be considered local to the macro, or known only within the macro definition. The list of formal and local parameters are identified at definition time and replaced with the parameter markers (character 77) so that the names of the substitutable arguments (formal and local) need not be retained after definition time. If a substitutable parameter name appears in the LOCAL list, it is ignored. The total number of local symbols plus substitutable parameters may not exceed 63. For each local symbol defined within the macro, the assembler creates a symbol and substitutes it for each use of the declared symbol. The created symbols appear as ↑↓ nnnnnn, where n is unique for each local symbol in a subprogram. The symbol A, for example, if it is declared local to the macro, may co-exist with another symbol A defined elsewhere in the subprogram.

Created symbols are substituted for local symbols wherever they appear in the macro except on comment lines with an * in column 1. Created symbols are not listed in the symbol reference table. Blanks are preserved in created symbol substitution; COMPASS makes no attempt to compress the line.

All symbols defined within the macro which are not local are global. Global symbols are accessible outside the macro definition, but local symbols are not.

A local symbol may be passed to inner macro definitions or inner macro calls.

Example:

```
ABC     MACRO    A,B
        LOCAL    C
C       BSS      10
          .
          .
XYZ     MACRO    D
        SA1      C
          .
          .
        ENDM
```

If the representation of C is ↑↓000010, when COMPASS defines the macro XYZ (when ABC is called), it is as if the definition were:

```
XYZ     MACRO    D
        SA1      ↑↓000010
          .
          .
        ENDM
```

Note the difference, however, between the above examples and the following:

```
ABC    MACRO   A,B
       LOCAL   C
C      BSS     10
         .
         .
         .
XYZ    MACRO   D
       LOCAL   C
       SA1     C
         .
         .
         .
       ENDM
```

When XYZ is defined, it appears as follows to COMPASS:

```
XYZ    MACRO   D
       LOCAL   ⇅000010
       SA1     ⇅000010
         .
         .
         .
       ENDM
```

The symbol ⇅000010 will be replaced with another invented symbol, and the reference to C in the SA1 instruction will not result in a reference to the C of the outer macro.

Thus, like substitutable parameters, invented symbols will replace LOCAL-named symbols wherever they appear in a macro definition, including inner macro definitions and inner macro calls.

6.1.2 MACRO BODY

Following MACRO, the first line which is not a LOCAL statement or a comment is the start of the macro body. The macro body consists of a series of symbolic instructions. Within these lines, in any field, may appear the name of a substitutable parameter listed on the MACRO line. To be recognized as such, the parameter must be bounded by two of the following characters:

= . ≠ - * / $, →) (Δ

Beginning of statement (column 1 or 2) or end of statement is also a delimiter. The character → may be used to catenate a substitutable parameter name with some other item, or to flag a parameter name not bounded by any other special characters and might not otherwise be recognized. Each → in the definition is removed when the macro is called, and the items it connects are catenated.

For example, if the parameter P1 is substituted in the expansion by A2, and P2 by A, then

```
S → P1     P1+1R → P2     becomes     SA2     A2+1RA
```

As this example indicates, the substitutable parameters may appear in any field of a statement in the macro body. However parameters are ignored in a comment line with an * in column 1. Likewise, comment cards within a macro definition are ignored and not reproduced when the macro is called.

Any instructions, except END, including other macro definitions and/or macro calls, may appear within a macro definition. Macro definitions appearing within another macro definition are not defined by COMPASS until the outer macro is called; therefore, inner macros may not be called before the outer macro is called, and must be called according to general macro rules.

Example:

```
          .
          .
          .
NAME1 MACRO  A
      SB1    A
NAME2 MACRO  A
      SB4    A
NAME2 ENDM
      NAME2  ALPHA     NAME2 is a valid call since it is not
          .            recognized as a macro call until
          .            NAME1 has been called and expanded.
NAME1 ENDM
          .            NAME2 may not be called in this
          .            part of the subprogram.
      NAME1  X
      NAME2  X         NAME2 is a valid call since NAME1
          .            has been called already.
          .
```

Since the characters = . $) (act as delimiters in the macro body for formal parameters, the programmer must be careful if he uses these characters in symbols. For example, given the macro definition:

```
ABC   MACRO   Z,VAL
Z     SET     VAL
      SA7     Z.ALPHA
        .
        .
      ENDM
```

and the macro call:

```
ABC   IOTA,1
```

The reference in the SA7 instruction is not to the symbol Z.ALPHA but to IOTA.ALPHA and is illegal since the symbol name is too long. The entire expansion is:

```
IOTA  SET     1
      SA7     IOTA.ALPHA
```

6.1.3 MACRO TERMINATOR

ENDM terminates a macro definition.

Location	Operation	Variable
blank or macro name	ENDM	ignored

To be recognized as a macro definition terminator, the ENDM location field must be blank or contain the name of a macro being defined. An ENDM with a blank location field terminates any and all macros being defined; a named ENDM terminates a macro with the same name together with its inner macros. An ENDM which terminates a definition also terminates any inner macro definitions for which a matching ENDM was not found.

6.2 MACRO CALL

A macro name in an operation field constitutes a macro call; it may contain a symbol in the location field, and a parameter list in the variable field. The parameter list of the macro call is scanned to identify and extract the character strings to be substituted for parameters of the macro definition. The parameter list has the following form:

p,p,p,...,p

p is a character string denoting an actual parameter. p may contain any characters except blank or , which are allowed only when enclosed within parentheses.

Parameters of the macro call are listed in the same order as the formal parameters in the macro definition. Missing actual parameters are empty, or null, and extra actual parameters are discarded. An explicit zero, if desired, must be entered as a parameter. A blank terminates the parameter list unless the blank is contained within parentheses.

When the left parenthesis is the first character of any parameter, all characters between it and the matching right parenthesis are considered part of that parameter. The outer pair of parentheses is removed when the parameter is substituted in a line. Parenthesized items may be embedded provided parentheses are properly paired. Parenthesized items may contain blanks and commas.

Example: If the macro XAM is defined:

```
XAM   MACRO   A,B
      LDM     A
      LJM     B
      ENDM
```

and a call is issued:

```
      XAM     (SUM,10B),(SAM,IND3)
```

COMPASS will expand the call as:

```
      LDM     SUM,10B
      LJM     SAM,IND3
```

Using the same macro XAM but with a call:

```
      XAM     SUM,SAM
```

COMPASS will expand the call as:

```
      LDM     SUM
      LJM     SAM
```

Processing of a location symbol on the macro call is dependent on the way the macro was defined:

<u>Standard Form Macro Definition</u> (macro name appeared in the location field):

A location symbol on the macro call line causes a force upper and the symbol is defined as the value of the location counter. For example, if the macro XAM is defined:

```
XAM   MACRO   A,B,C
      SB1     A
      SB2     B+C
      ENDM
```

and a call is issued:

```
LOC   XAM     X,Y
```

COMPASS expands the call as if it were:

```
LOC   BSS     0
      SB1     X
      SB2     Y
```

If, however, there is no location symbol on the call, no force upper occurs and the SB1 operation falls into the first available space.

Alternate Form Macro Definition (macro name appeared as the first variable field subfield):

The location symbol of the macro call is passed as the actual parameter to be substituted for the first formal parameter (the location argument) in the definition. Forcing upper is determined by the first instruction of the expansion. If there is no location field symbol on the macro call, the first argument is null or blank.

For example, if macro XAM is defined:

```
        MACRO   XAM,A,B,C
A       SB1     B
        SB2     C
        ENDM
```

and a call is issued:

```
LOC     XAM     X,Y
```

the expansion appears as:

```
LOC     SB1     X
        SB2     Y
```

A force upper occurs because of the location field entry in the first line. If, however, macro XAM is defined:

```
        MACRO   XAM,A,B,C
        SB1     B
A       SB2     C
        ENDM
```

and a call is issued:

```
LOC     XAM     X,Y
```

the expansion appears as:

```
        SB1     X
LOC     SB2     Y
```

No force upper occurs for the SB1 operation but it does occur for the SB2.

Also if the macro XAM is defined:

```
        MACRO   XAM,A,B,C
A       SB1     B
        SB2     C
        ENDM
```

and a call is issued:

```
        XAM     X,Y
```

then the expansion appears as:

```
        SB1     X
        SB2     Y
```

No force upper occurs since parameter A is null.

6.3 OPDEF

The OPDEF macro permits definition or redefinition of instructions in the COMPASS format of central processor machine instructions; the macro call is written in the same format as central processor operations. OPDEF provides more extensive control than the standard macro form.

6.3.1 OPDEF DEFINITION

The pseudo instruction OPDEF is used in place of MACRO. The OPDEF heading line is followed by the macro definition (if needed), and ENDM specified in the manner described for MACRO.

The OPDEF heading line indicates the mnemonic name and variable field format which are recognized as an OPDEF call, and lists the substitutable parameters as follows:

Location	Operation	Variable
Description of operation field and variable field of the OPDEF call	OPDEF	parameter list

Location Field of the OPDEF Line

This field contains an abbreviated description of the entire instruction to be recognized as an OPDEF call, including operation code, registers and/or address expressions which constitute the variable field, and subfield separators of the variable field in the macro call.

The first part of the location field entry describes the operation field of the OPDEF call; it consists of two letters. The first may be any letter; the second may be a register designator: A, B, or X. In this case, the operation field of the OPDEF macro call is defined to be aAn, aXn, or aBn.

a = a unique identifier

n = 0-7

If the second letter is not A, B, or X, the operation field of the OPDEF macro call is defined as a two-letter mnemonic, such as EQ.

The second part of the location field entry describes the variable field of the OPDEF call. It includes all registers and/or address expressions which constitute the variable field as well as all subfield separators. This part of the OPDEF name may contain none, one, two, or three of the following 22 subfield descriptors, each descriptor separated by a comma; r represents a register letter, A, B, or X; Q represents an address expression.

void	Q
r	rQ
-r	-rQ
r+r	r+rQ
-r+r	-r+rQ
r*r	r*rQ
-r*r	-r*rQ
r/r	r/rQ
-r/r	-r/rQ
r-r	r-rQ
-r-r	-r-rQ

For example, -r*r could describe -X3*X0; rQ could describe B2+ALPHA.

The two parts of the OPDEF location field — op code description and variable field descriptors — are not separated by a special character unless this character is the operator of the first descriptor. Examples of the OPDEF name field (location field of the OPDEF line) and the macro call described are as follows:

Name Field	Call Described
Single descriptor, of the form Q	
JPQ	JP address expression
Single descriptor of the form rQ	
JPBQ	JP Bn±address expression
Single descriptor of the form r+rQ	
JPB+BQ	JP Bn±Bn±address expression
Three descriptors of the form r, r, and Q	
NEB, B, Q	NE Bn, Bn, address expression
Three descriptors of the forms r-r, r-r, and Q	
LJB-B, A-X, Q	LJ Bn-Bn, An-Xn, address expression
One descriptor of the form -r*r	
BX-X*X	BXn -Xn*Xn
Single descriptor of the form r+r	
SBX+B	SBn Xn+Bn
Two descriptors of the forms r and r	
LXB, X	LXn Bn, Xn

In the OPDEF call, an address expression must be preceded by a plus or minus unless the Q in the descriptor is not combined with register letters.

Examples:

OPDEF Name Field	Call
JPQ	JP address expression
JPBQ	JP Bn±address expression
JPB, Q	JP Bn, address expression
JPX/XQ	JP Xn/Xn±address expression

In the following examples of OPDEF location field entries, all instructions have been made to resemble legal COMPASS machine mnemonics.

To identify the JP instruction with a single address expression	JPQ
To identify JP Bj+K	JPB+Q
To identify NE Bj, Bk, K	NEB, B, Q

To identify Bxi -Xk*Xj	BX-X*X
To identify SBi Xj+Bk	SBX+B
To identify SBi Bj+Xk	SBB+X

Operation Field of OPDEF Line

OPDEF

Variable Field of OPDEF Line

parameter list

The number of formal parameters listed in the OPDEF instruction variable field must match the total number of register and expression designators (A, B, X, and Q) in the parameter list and must appear in the same order. Parameters may be separated by any of the characters

, + - * / .) ($ =

The list is terminated by a blank.

Examples of Complete OPDEF Definitions

To redefine the single-address long jump, JP, as the fast jump, EQ;

```
JPQ     OPDEF   P1
        EQ      P1
        ENDM
```

All JP instructions subsequently encountered which match the format described by the OPDEF location field are expanded as EQ. JP instructions not of that format, such as JP B3+ALPHA, are not effected.

To trap all floating double precision subtraction instructions (DXi Xj-Xk) and jump to an error check routine for debugging: I, J, and K are substitutable parameters used within the definition prototype.

```
DXX-X   OPDEF   I,J,K
          .
          .
          .
        RJ      CKOUT
        ENDM
```

To define a new instruction as a set of code which performs a complete integer divide each time it is called and expanded:

```
IXX/X   OPDEF   P1,P2,P3
          .
          .
        integer divide code
          .
          .
        ENDM
```

Each time an instruction of the format IXn Xn/Xn is used, the macro is expanded.

To define RXi k to be the same as AXi k

```
RXQ     OPDEF   P1,P2
        AX.P1   P2
        ENDM
```

The instruction RXi Xj $\begin{bmatrix} + \\ - \\ * \\ / \end{bmatrix}$ Xk are not effected.

6.3.2
OPDEF CALLS

The registers and/or address expressions used in the macro call must match exactly the number and order of registers and/or expressions indicated in the OPDEF location field description or the line is not considered an OPDEF macro call. For example, given the definition header:

```
SXX+B   OPDEF   I,J,K
```

The following lines do not cause an expansion of the macro:

```
SX5     X4
SX5     B3+X4
SX5     B3
```

Only a line of the format SXn Xn+Bn causes an expansion.

Location field entries on an OPDEF or MACRO-defined call are equivalent on a normal MACRO-defined macro call.

OPDEF definitions may appear anywhere in a subprogram, OPDEF calls are recognized at any place after the definition.

OPDEF-defined and MACRO-defined macros differ in the following characteristics:

- Unlike MACRO-defined macros, only the register value given in the call of an OPDEF-defined macro is used in the substitution of parameters. For example, using the IXX/X macro illustrated above, the following code might be included in its definition:

```
IXX/X   OPDEF   P1,P2,P3
        PX.P2   X.P2
        PX.P3   X.P3
        NX.P2   X.P2,B4
          .
          .
        ENDM
```

  The instruction which calls the IXX/X macro might be:

```
        IX3     X4/X.DIV
```

  The parameters passed along to the macro body are 3, 4, and DIV; X3, X4, and X.DIV are not passed along to the macro body.

- Actual parameters of an OPDEF call are separated by + - * / or comma according to the definition of the OPDEF macro; only the comma may be used to separate parameters of a MACRO-defined macro.

6.4 SYSTEM MACROS

Macros of such general usefulness that they should be available to any program without each program defining them may be defined as system macros; or they may be defined as a result of the XTEXT definitions contained on a separate file accessible to COMPASS.

System macros are defined by SCOPE for communication with the operating system. They include such system functions as opening and closing files, reading, writing, and specifying parameters for a file environment table. The definitions of these macros exist on a system-maintained file, and are available to COMPASS for every assembly. The programmer simply writes a macro call whenever a system macro is needed. Use of the system macros is detailed in SCOPE reference documents.† The file of systems text may

---

† 6400/6500/6600 SCOPE 3.1 Reference Manual, Publication number 60189400.

contain any kind of legal macro definition, including OPDEF. The system macro definitions are not included in the subprogram listing. The expansion of a system macro call may be obtained by using the S option on the LIST pseudo instruction. System macros cannot redefine COMPASS mnemonics.

6.5 OPERATION CODE RECOGNITION ORDER

COMPASS interprets an operation code according to the following order of precedence:

1. Programmer macro (highest)
2. System macro
3. COMPASS machine or pseudo instruction (lowest)

The entry in operation code field is compared with the operation code table which contains all system and programmer defined MACRO names, all PP machine instructions, all COMPASS pseudo instructions (except IDENT and LOCAL). If the instruction or macro is contained in the operation code table, the operation has been identified. If no match is found and a CP assembly is in progress, a syntactic analysis of the entire address field and operation code is made. COMPASS attempts to match this entry with another table which includes all CP instructions and all system and programmer defined OPDEF macro names/descriptions. If the search fails to produce a match, an operation code error is issued.

With an OPDEF or MACRO definition, COMPASS searches the operation code table first for a match. For a MACRO definition, the macro name is used in the search. If a MACRO name matches any other name in the table, a duplicate definition flag is issued, and the new definition replaces the old one. For an OPDEF definition, the entry for the search is a descriptor of the same format as the CP machine and other OPDEF descriptions in the operation code table. If an OPDEF descriptor matches any other descriptor in the table, a like replacement occurs. OPDEF descriptors do not match any name in the table: an OPDEF will not redefine a MACRO name, a PP machine instruction mnemonic, or a pseudo instruction name. Conversely, a MACRO name will not match any of the OPDEF or CP mnemonic descriptors in the table: a MACRO will not cause duplicate definition of any OPDEF defined macro or CP mnemonic. A duplicate macro definition flag is produced when a macro name is the same as a previous macro name (system or programmer defined), a PP machine instruction (if PP assembly), or a pseudo instruction.

A duplicate definition flag is produced also when an OPDEF name/description is the same as a CP instruction or a previous OPDEF name/description (system or programmer defined). A MACRO definition SB4 will redefine the machine instruction SB4 but only because the SB4 macro exists at the same time as the description of all other SBx or SB.x CP instructions. The entry SB4 in the operation code table will be found before COMPASS tries the syntactic analysis to find a CP mnemonic.

For example, if a macro named SX5 is defined, duplicate definition of other SXn CP instructions does not result. If a later OPDEF definition occurs which redefines all instructions of the form SXr+r, a duplicate definition of all other SXn rn+rn instructions results and the duplicate definition flag is issued. Thereafter, if a SX5 instruction is encountered, the SX5 macro is expanded since it has not been redefined. However, if a SXm rn+rn instruction is encountered where m is not 5 the OPDEF definition will be expanded since all instructions of the format SXn rn+rn were redefined by the OPDEF.

MICROS 7

The COMPASS micro capability enables the programmer to reference symbolically a defined character string. Use of a micro definition requires two steps: defining the character string and substituting the micro. At assembly time, the defined character string is substituted at any point in the line where the micro name appears prior to any other interpretation of the statement.

7.1 MICRO SUBSTITUTION

At any place in a statement a micro mark (≠) may appear followed by a string of characters and another micro mark. The intervening characters constitute a micro name and signal a micro substitution is to be made at that point.

Example: The micro NAME might be defined as the characters

```
LOC SA1 ADDRESS+
```

then, a symbolic instruction introduced as follows, in column 2

```
≠NAME≠4
```

would be changed by COMPASS into

```
LOC SA1 ADDRESS+4
```

where LOC begins in column 2.

If the second micro mark does not appear or if the micro name is unknown, a non-fatal assembly error results and no substitution is made. Micro marks are not processed if they appear in comment lines (* in column 1), but they are processed if they are written in the comment field of an instruction line.

If, as a result of micro substitution, column 72 of the last card read is exceeded, the assembler creates continuation cards up to a maximum of 9. Any excess is discarded without comment.

7.2 MICRO DEFINITION

The MICRO pseudo instruction is used to define a character string and to assign a name to that micro string.

Location	Operation	Variable
micro name	MICRO	3 subfields separated by commas

The variable field subfields are, in order:

Absolute address expression n_1

Absolute address expression n_2

Delimited character string, dccc...ccd. The delimiter d is any character, and ccc...cc is a string of any characters other than character d.

Counting the first character after d as character 1, the string is formed by extracting n_2 characters starting with character n_1. For example:

```
NAME      MICRO      1,19,*ALPHANUMERIC STRING*
```

If the second delimiter occurs before count n_2 is exhausted, the string is terminated at that point. If n_1 is non zero, and n_2 is zero or absent, the character string is considered to include all characters between character n_1 and the closing delimiter. The following example is therefore equivalent to the above.

```
NAME      MICRO      1,,*ALPHANUMERIC STRING*
```

If n_1 is zero or absent, the character string is empty, and no substitution takes place when this micro name is given in an instruction line. n_2 and the character string are ignored.

Previously defined micros may appear as part of a micro definition; one micro may be defined as a substring of another. For example, assuming the micro

```
NAME1     MICRO      1,25,*MAJOR ALPHANUMERIC STRING*
```

has been defined in the program, an equivalent micro to the examples above can be achieved by the micro:

```
NAME      MICRO      7,,*≠NAME1≠*
```

Also a micro may be defined as a combination of multiple, previously defined micros. The following series would result in another equivalent to the previous examples:

```
NAME1     MICRO     1,12,*ALPHANUMERIC*
NAME2     MICRO     1,7,*ΔSTRING*
NAME      MICRO     1,,*≠NAME1≠≠NAME2≠*
```

The delimiter (* in the example) may not appear in either of the character strings substituted for NAME1 or NAME2. If the delimiter is encountered before the count n_2 is satisfied, the string will be ended.

A micro may be redefined; NAME may be originally defined as one character string and subsequently defined, with a different character string. After the redefinition, the original character string is no longer known to the assembler. The original micro may also be used as part of the redefinition.

Example:

```
NAME     MICRO     1,6,*STRING*
          .
          .
          .
         series of statements (A)
          .
          .
          .
NAME     MICRO     1,19,*ALPHANUMERIC≠NAME≠*
          .
          .
          .
         series of statements (B)
          .
          .
```

During statement series A the first definition of NAME prevails. During statement series B the redefinition of NAME prevails and the original string no longer exists.

Micros of different names but with identical character strings may co-exist at any time. Varied manipulation of character strings — testing for a particular character, counting characters, catenating strings, etc. — is possible in COMPASS with the use of MICRO in conjunction with IFC, DUP, STOPDUP, and SET pseudo instructions.

ASSEMBLER INPUT/OUTPUT

8.1 COMPASS CONTROL CARD

The files COMPASS uses are specified on the control card:

COMPASS(L=fname,I=fname,B=fname,S=rname or SCPTEXT)

The specifications may be in any order; the characters = , (may be used interchangeably as separators; the characters . and) are card terminators. L, I, B and S may not be used as file names.

Each option is specified as follows:

L option:	absent	Full listings on OUTPUT
	L	Full listings on OUTPUT
	L=0	Brief listings on OUTPUT
	L=fname	Full listings on file fname
I option:	absent	Input from INPUT
	I	Input from INPUT
	I=fname	Input from file fname
B option:	absent	Binary on LGO
	B	Binary on LGO
	B=0	Suppress binary
	B=fname	Binary on file fname
S option:	absent	Systems text from SYSTEXT
	S	Systems text from SYSTEXT
	S=rname	Systems text from library overlay named rname
	S=SCPTEXT	Systems text from library overlay named SCPTEXT which contains the system symbols definitions.

8.2 INPUT AND OUTPUT FILES

COMPASS assembles all statements beginning at the current position of the input file until an end-of-record or end-of-file. If the input file is positioned at an end-of-file mark (file is empty), COMPASS produces a fatal error.

Other input is from the system text record and XTEXT files. All input cards may be 90 columns; longer cards are truncated. All input files are coded. The assembly output consists of one logical record of listable output for 136-column printers, and several logical records of binary output.

Scratch File

For large assemblies, a magnetic tape scratch file may be used to eliminate disk conflicts. Use of a magnetic tape scratch file has a negligible effect upon CP time, but improves throughput time considerably. This may be accomplished by assigning a file named CMPSCR to tape, or a scratch file may be maintained in mass storage. Care must be taken with the use of a scratch file; it must be re-read by COMPASS exactly as written. If a write or read error occurs, it should not be bypassed; the job should be restarted.

8.3 FIELD LENGTH REQUIREMENTS

All COMPASS tables are variable; it is not possible to specify an exact field length. For most assemblies, a field length of 34000_8 should be sufficient. As part of the listable output, COMPASS gives the amount of storage not needed for the assembly. The field length can then be decreased for subsequent runs.

When COMPASS does not have enough storage to complete processing, part or all of the reference table is discarded. If this fails to release enough storage, assembly terminates with a dayfile message.

8.4 LISTABLE OUTPUT

COMPASS list output contains as minimum header information: program name and length, block names and length, external symbol names, entry points. In addition, any lines which cause an error flag to appear are unconditionally listed. At the end of assembly, an error directory and assembler statistics appear.

8.4.1 HEADER INFORMATION

At the beginning of the listing, all blocks are listed as shown below (all programmer defined blocks, even zero length, are listed).

Origin	Length	Name	Type
nnnnnn	nnnnnn	ABSOLUTE*	local
nnnnnn	nnnnnn	PROGRAM*	local
nnnnnn	nnnnnn	LITERALS*	local
nnnnnn	nnnnnn	$NAME_1$	local or common
⋮	⋮	⋮	⋮
⋮	⋮	(programmer-declared blocks)	
⋮	⋮	⋮	⋮
nnnnnn	nnnnnn	$NAME_n$	local or common

8.4.2 ASSEMBLED CODE

The LIST pseudo instruction specifies the contents of the listing; however, the COMPASS control card provides an external list control which overrides any LIST directives. If the external option to list is not selected (L=0) only header information and error diagnostics are listed; if the external option to list is selected, listing control is directed by the internal LIST options.

Each line of the listing will contain the following items after the header information:

- Error flags, if any
- LOC flag (an L if location counter is different from origin counter)
- Location counter value
- Octal value of code
- Address relocation indicator
- Card image (columns 1-72)
- Columns 73-90 of the source line, or an indication of source if generated line

8.4.3 DIAGNOSTICS, REFERENCE TABLE, AND STATISTICS

Errors detected by COMPASS are fatal or non-fatal. Any fatal error will suppress binary output as well as terminate the job when assembly is finished. Non-fatal errors are merely warnings. Errors flagged with an alphabetic character are fatal; non-fatal warning flags are numeric. All lines with errors are listed. A one-character indication of each error on the line appears to the left. At the end of the assembly an error directory is listed. The pages on which each error occurred are noted, and a brief description of the error is given.

<u>FATAL ERROR FLAGS</u>

L Location field bad. Occurs only on instructions which require a location field entry. Illegal entries in other location fields produce a non-fatal error flag since the illegality might not affect the rest of the assembly.

O Operation field bad:

- Unrecognized entry in the operation field
- Operation and address fields do not describe a valid CP instruction
- Unrecognized modifier in IF or IFC
- Operation not in correct place, such as ABS or PERIPH

A Address field bad. A general flag indicating an illegality in the variable field. Can occur on any operation.

D Doubly defined symbol. Appears on all operations which attempt to define a symbol with a value different than its previous value.

R Data origins outside block; data is loaded outside the block ranges, or into blank common.

F Number of entries exceeds permissible amount:

   Total number of words required for any one literal, data item, or the entire address field of a LIT operation exceeds 100

   More than 63 parameters appear in a macro definition

   Assembler symbol table limit exceeded. This limit is 4096-4350 depending upon the symbols used

U An undefined symbol is referenced. The value of the symbol and the expression in which it appears are set to zero.

V Invalid bit count on a VFD instruction. It must be an absolute value between 0 and 60.

P Produced by an ERR instruction.

S Segment error, word count zero.

NON-FATAL ERROR FLAGS

1 Bad location field entry. The symbol will not be defined.

2 Bad address element on a symbol definition instruction. The location symbol will not be defined.

3 Macro redefines a previously known operation.

4 Bad parameter name is ignored.

5 OPDEF is incorrectly specified.

6 Location field is meaningless.

7 Address value exceeds field size; the result is truncated.

8 Address subfield is missing, or there are too many subfields.

9 Micro substitution error, no substitution will be made; or attempt was made to use a semicolon in a source statement.

Following the error directory the assembler statistics are listed:

Decimal count of statements processed by COMPASS, including all generated lines

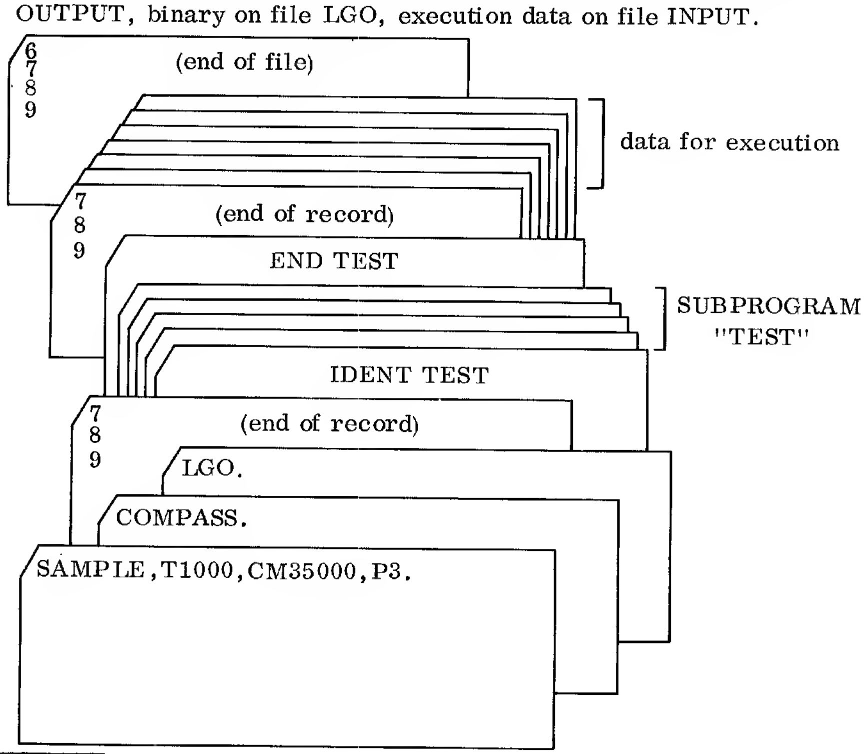
Indication of storage unused by the assembler which permits adjustment of field length in subsequent assemblies

Decimal count of reference table entries discarded because of restricted storage, if any

If a symbol reference table is requested, it is listed next. The reference table contains all symbols in alphabetical order (sorted according to the collating sequence in Appendix A), with their relocation value, and all reference locations. Undefined symbols also appear, with a U error.

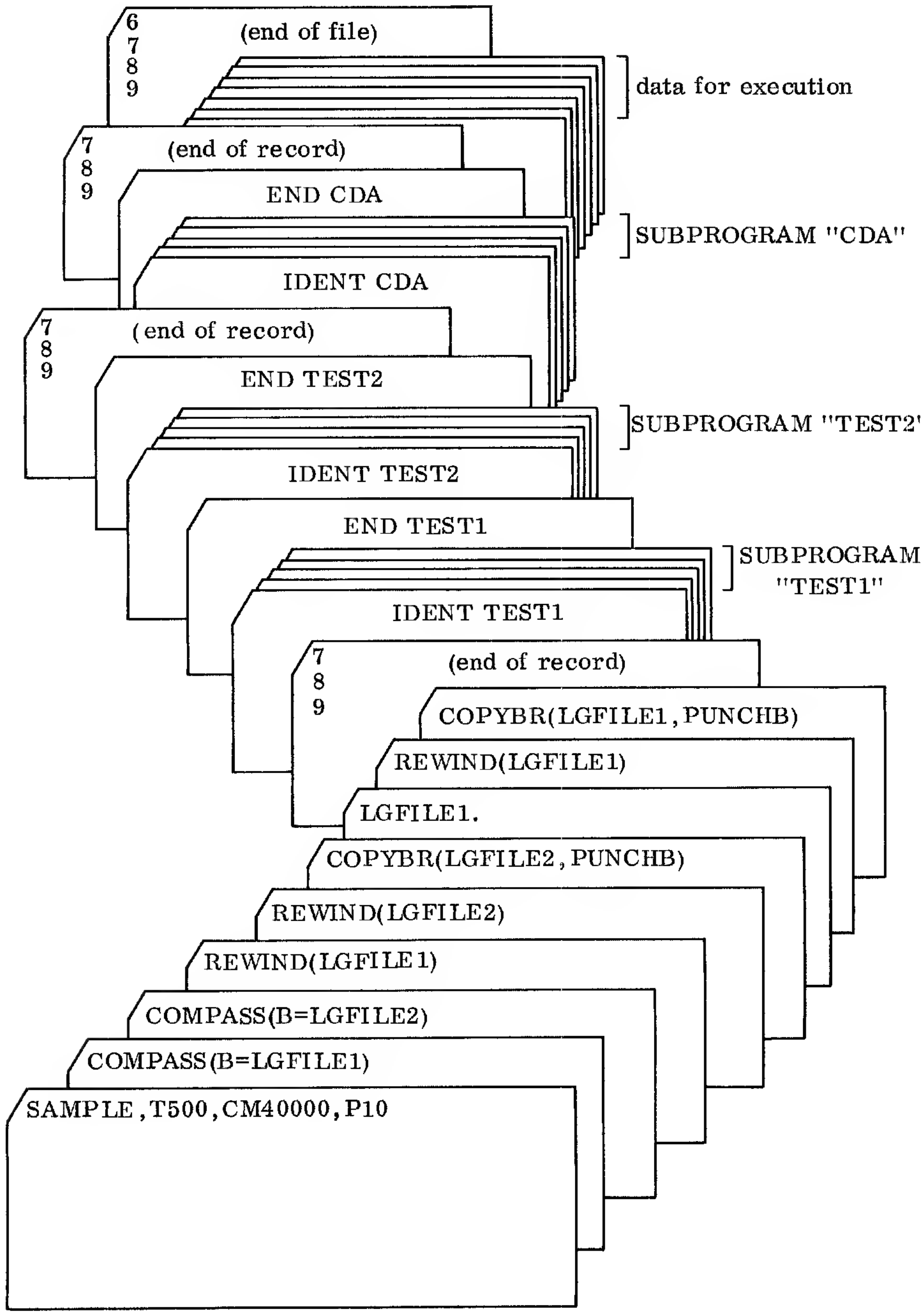
8.5 EXAMPLES OF JOBS

(1) Assembly with listing and binary output; subprogram execution with data input. Source logical record is on file INPUT, listing on file OUTPUT, binary on file LGO, execution data on file INPUT.

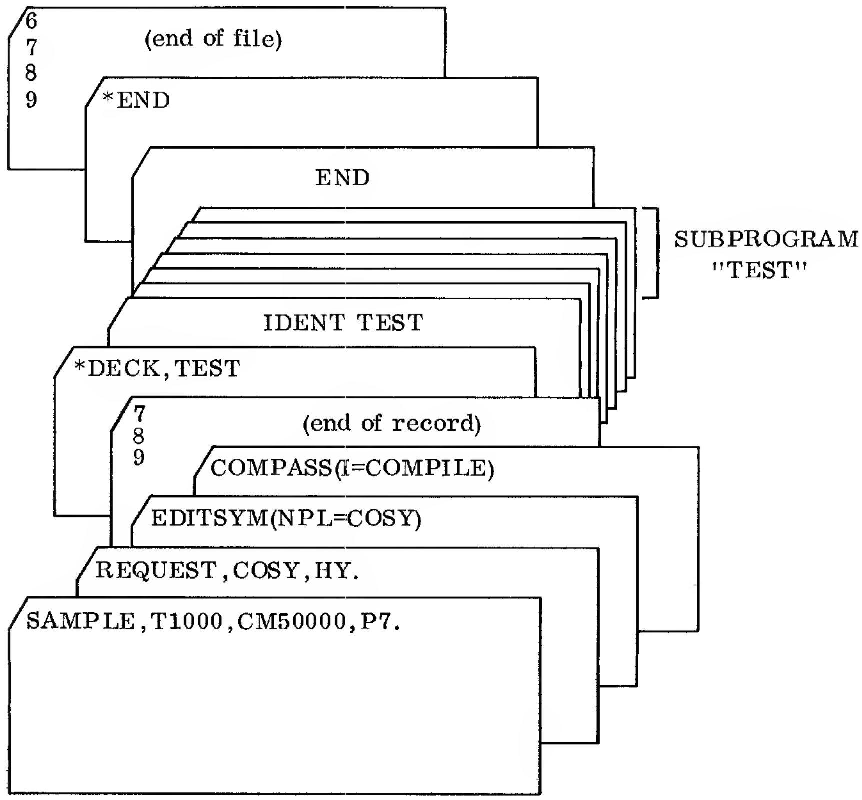


† In the examples SCOPE operating system control cards have been included. For description of their parameters and use, see the 6400/6500/6600 SCOPE 3.1 Reference Manual (Pub. No. 60189400).

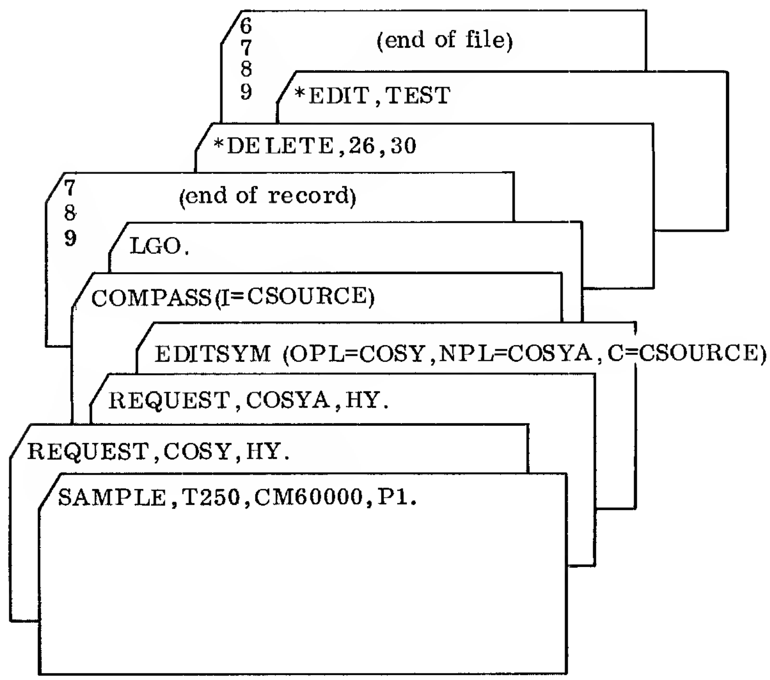
(2) Batch assemble with listing and binary output; punch the binary output and execute the first program.



(3) Create a compressed symbolic deck (via EDITSYM) of a subprogram. Assemble with listing.



(4) Update the compressed symbolic record COSY created in the previous record; write corrected compressed record on file COSYA and a corrected source record on file CSOURCE. Assemble the file CSOURCE and execute.



APPENDIX SECTION

CHARACTER CODES COLLATING SEQUENCE A

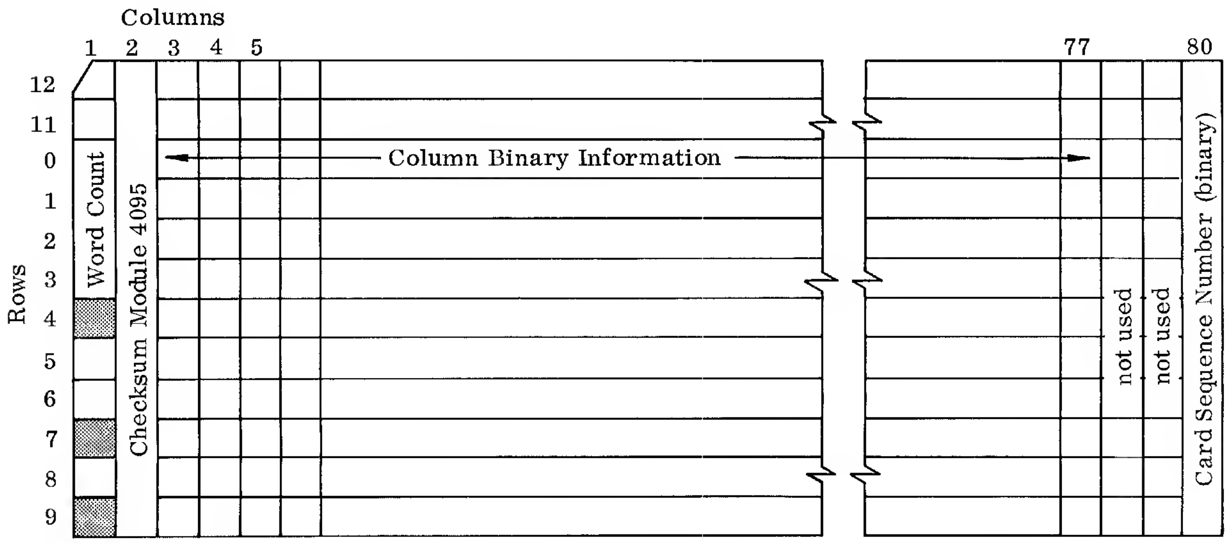
Character	Display Code	External BCD	Hollerith Punch Positions
A	01	61	12-1
B	02	62	12-2
C	03	63	12-3
D	04	64	12-4
E	05	65	12-5
F	06	66	12-6
G	07	67	12-7
H	10	70	12-8
I	11	71	12-9
J	12	41	11-1
K	13	42	11-2
L	14	43	11-3
M	15	44	11-4
N	16	45	11-5
O	17	46	11-6
P	20	47	11-7
Q	21	50	11-8
R	22	51	11-9
S	23	22	0-2
T	24	23	0-3
U	25	24	0-4
V	26	25	0-5
W	27	26	0-6
X	30	27	0-7
Y	31	30	0-8
Z	32	31	0-9
0	33	12	0
1	34	01	1

Character	Display Code	External BCD	Hollerith Punch Punch
2	35	02	2
3	36	03	3
4	37	04	4
5	40	05	5
6	41	06	6
7	42	07	7
8	43	10	8
9	44	11	9
+	45	60	12
-	46	40	11
*	47	54	11-8-4
/	50	21	0-1
(	51	34	0-8-4
)	52	74	12-8-4
$	53	53	11-8-3
=	54	13	8-3
blank	55	20	space
,	56	33	0-8-3
.	57	73	12-8-3
≡	60	36	0-8-6
[	61	17	8-7
]	62	32	0-8-2
:	63	00	8-2
≠	64	14	8-4
→	65	35	0-8-5
∨	66	52	11-0
∧	67	37	0-8-7
↑	70	55	11-8-5
↓	71	56	11-8-6
<	72	72	12-0
>	73	57	11-8-7
≤	74	15	8-5
≥	75	75	12-8-5
¬	76	76	12-8-6

CARD FORMAT B

Column 1

7,8,9	End of logical record
6,7,8,9	End of file
7,9	Binary card
7 and 9 not both in column 1	Coded card



A binary card can contain up to 15 central memory words starting at column 3. Column 1 also contains a central memory word count in rows 0, 1, 2 and 3 plus a check indicator in row 4. If row 4 of column 1 is zero, column 2 is used as a checksum for the card on input; if row 4 is one, no check is performed on input.

Columns 78 and 79 of a binary card are not used, and column 80 contains a binary serial number. If a logical record is output on the card punch, each card has a checksum in column 2 and a serial number in column 80, which orders it within the logical record.

Coded cards are translated on input from Hollerith to display code, and packed 10 columns per central memory word. A central memory word with a lowest byte of zero marks the end of a coded card (it is a coded record), and the full length of the card is not stored if it has trailing blanks. A compact form is thereby produced if coded cards are transferred to another device.

Card Files

Any punched cards can be read: standard types or free-form cards.

Four types of cards are considered standard:

A card with 0017 octal in column 1 is recognized as an end-of-file marker.

A card with 0007 octal in column 1 is recognized as an end-of-record marker. The level is assumed to be zero unless columns 2 and 3 contain a level number punched in Hollerith form. The level number is read as octal. The following are valid punches (b represents a blank):

00 or 0b	04 or 4b	10	14
01 or 1b	05 or 5b	11	15
02 or 2b	06 or 6b	12	16
03 or 3b	07 or 7b	13	17

Any card other than the above with 7, 9 punches in column 1 is assumed to be binary. It must contain 0105, 0205, 0305....... 1605, or 1705 in column 1 and a correct checksum in column 2; or 0145, 0245...... 1645, or 1745 in column 1, in which case column 2 is ignored. The first two digits, 01 or 17, give the word count of the card. Each word occupies 5 columns, and the first word of information begins in column 3. Columns after the last word of information, up to and including column 78, are ignored. The lower 5 bits of column 79, and all 12 bits of column 80 constitute a 17-bit serial number for the card within its record. If the cards of a binary record do not have these numbers in correct sequence (beginning at 1 for the first card), a message is given but the cards are accepted. The checksum is the one's complement of the sum of all information columns; this sum is formed as if in a 12-bit accumulator with circular carry.

Any card that does not have 7 and 9 punched in column 1 is assumed to contain Hollerith-punched information, one 6-bit character per column, or eight 60-bit words per card. Any column that does not contain a valid Hollerith combination is read as a blank, and a message containing the record number and the card number within the record is given. To be a valid Hollerith combination, a column must contain one of the following:

12 and 0, or 11 and 0, and no other punches

or

Not more than one of the punches 12, 11, and 0, with

No additional punch, or any one punch from 1 to 9

or

An 8 punch with one more punch from 2, 3, 4, 5, 6, 7

Binary and Hollerith-punched (coded) cards may be mixed within one record, but a message is given containing the number of any record containing one or more mode changes.

CENTRAL PROCESSOR MNEMONICS C

Instructions are listed in order of octal operation value. In the operation field and variable field subfield notations, the following symbology is used:

A, B, X	register symbols	i, j, k	register number
K	address expression (18 bits)	n	absolute address (6 bits)

Octal	Mnemonic	Variable Field	Length (bits)	Page
0000000000	PS		30	4-12
0100k	RJ	K	30	4-12
011jk	RE	Bj+K	30	4-25
012jk	WE	Bj+K	30	4-26
0130000000 4600046000	XJ	Bj+K	60	4-12
02i0k	JP	Bi+K	30	4-13
030jk	ZR	Xj, K	30	4-13
031jk	NZ	Xj, K	30	4-13
032jk	PL	Xj, K	30	4-13
033jk	NG	Xj, K	30	4-13
034jk	IR	Xj, K	30	4-14
035jk	OR	Xj, K	30	4-14
036jk	DF	Xj, K	30	4-14
037jk	ID	Xj, K	30	4-14
0400k	ZR	K	30	4-14
0400k	EQ	K	30	4-14
04i0k	EQ	Bi, K	30	4-14
04i0k	ZR	Bi, K	30	4-14
04ijk	EQ	Bi, Bj, K	30	4-14
05i0k	NZ	Bi, K	30	4-15
05i0k	NE	Bi, K	30	4-15
05ijk	NE	Bi, Bj, K	30	4-15

Octal	Mnemonic	Variable Field	Length (bits)	Page
060jk	LE	Bj, K	30	4-15
06i0k	PL	Bi, K	30	4-15
06i0k	GE	Bi, K	30	4-15
06ijk	GE	Bi, Bj, K	30	4-15
06ijk	LE	Bj, Bi, K	30	4-15
070jk	GT	Bj, K	30	4-15
07i0k	LT	Bi, K	30	4-15
07i0k	NG	Bi, K	30	4-16
07ijk	LT	Bi, Bj, K	30	4-16
07ijk	GT	Bj, Bi, K	30	4-16
10ijj	BXi	Xj	15	4-16
11ijk	BXi	Xj*Xk	15	4-16
12ijk	BXi	Xj+Xk	15	4-16
13ijk	BXi	Xj-Xk	15	4-17
14ikk	BXi	-Xk	15	4-17
15ijk	BXi	-Xk*Xj	15	4-17
16ijk	BXi	-Xk+Xj	15	4-17
17ijk	BXi	-Xk-Xj	15	4-18
20ijk	LXi	jk	15	4-18
21ijk	AXi	jk	15	4-18
22i0k	LXi	Xk	15	4-18
22ijk	LXi	Bj, Xk	15	4-19
23ijk	AXi	Bj, Xk	15	4-19
24i0k	NXi	Xk	15	4-19
24ijk	NXi	Bj, Xk	15	4-19
25i0k	ZXi	Xk	15	4-20
25ijk	ZXi	Bj, Xk	15	4-20
26i0k	UXi	Xk	15	4-20
26ijk	UXi	Bj, Xk	15	4-20
27ijk	PXi	Bj, Xk	15	4-21
30ijk	FXi	Xj+Xk	15	4-21
31ijk	FXi	Xj-Xk	15	4-22

Octal	Mnemonic	Variable Field	Length (bits)	Page
32ijk	DXi	Xj+Xk	15	4-22
33ijk	DXi	Xj-Xk	15	4-22
34ijk	RXi	Xj+Xk	15	4-22
35ijk	RXi	Xj-Xk	15	4-23
36ijk	IXi	Xj+Xk	15	4-23
37ijk	IXi	Xj-Xk	15	4-23
40ijk	FXi	Xj*Xk	15	4-24
41ijk	RXi	Xj*Xk	15	4-24
42ijk	DXi	Xj*Xk	15	4-24
43ijk	MXi	jk	15	4-21
44ijk	FXi	Xj/Xk	15	4-25
45ijk	RXi	Xj/Xk	15	4-25
46000	NO		15	4-8
47ikk	CXi	Xk	15	4-25
50ijk	SAi	Aj±K	30	4-8
51i0k	SAi	K	30	4-9
51ijk	SAi	Bj±K	30	4-9
52ijk	SAi	Xj±K	30	4-9
53ij0	SAi	Xj	15	4-9
53ijk	SAi	Xj+Bk	15	4-9
54ij0	SAi	Aj	15	4-9
54ijk	SAi	Aj+Bk	15	4-9
55ijk	SAi	Aj-Bk	15	4-10
56ij0	SAi	Bj	15	4-10
56ijk	SAi	Bj+Bk	15	4-10
57i0k	SAi	-Bk	15	4-10
57ijk	SAi	Bj-Bk	15	4-10
60ijk	SBi	Aj±K	30	4-10
61i0k	SBi	K	30	4-10
61ijk	SBi	Bj±K	30	4-10

Octal	Mnemonic	Variable Field	Length (bits)	Page
62ijk	SBi	Xj±K	30	4-10
63ij0	SBi	Xj	15	4-10
63ijk	SBi	Xj+Bk	15	4-10
64ij0	SBi	Aj	15	4-11
64ijk	SBi	Aj+Bk	15	4-11
65ijk	SBi	Aj-Bk	15	4-11
66ij0	SBi	Bj	15	4-11
66ijk	SBi	Bj+Bk	15	4-11
67i0k	SBi	-Bk	15	4-11
67ijk	SBi	Bj-Bk	15	4-11
70ijk	SXi	Aj±K	30	4-11
71i0k	SXi	K	30	4-11
71ijk	SXi	Bj±K	30	4-11
72ijk	SXi	Xj±K	30	4-11
73ij0	SXi	Xj	15	4-11
73ijk	SXi	Xj+Bk	15	4-11
74ij0	SXi	Aj	15	4-11
74ijk	SXi	Aj+Bk	15	4-11
75ijk	SXi	Aj-Bk	15	4-12
76ij0	SXi	Bj	15	4-12
76i0k	SXi	-Bk	15	4-12
76ijk	SXi	Bj+Bk	15	4-12
77ijk	SXi	Bj-Bk	15	4-12

PERIPHERAL PROCESSOR MNEMONICS

Octal Value	Machine Instruction	Length (bits)	Page
0000††			
01dd mmmm	LJM m,d	24	4-35
02dd mmmm	RJM m,d	24	4-35
03rr	UJN r	12	4-34
04rr	ZJN r	12	4-34
05rr	NJN r	12	4-35
06rr	PJN r	12	4-35
07rr	MJN r	12	4-35
10rr	SHN r	12	4-30
11dd	LMN d	12	4-31
12dd	LPN d	12	4-31
13dd	SCN d	12	4-31
14dd	LDN d	12	4-28
15dd	LCN d	12	4-28
16dd	ADN d	12	4-29
17dd	SBN d	12	4-29
20cc cccc	LDC c	24	4-29
21cc cccc	ADC c	24	4-30
22cc cccc	LPC c	24	4-32
23cc cccc	LMC c	24	4-32
2400	PSN	12	4-28
2500††			

†Notations: c 18-bit address value
d 6-bit index value
m 12-bit address value
r number of steps to jump

††NOP instruction must be generated by data statement.

Octal Value		Machine Instruction		Length (bits)	Page
260d		EXN	d	12	4-35
261d		MXN	d	12	4-36
2700		RPN		12	4-36
30dd		LDD	d	12	4-28
31dd		ADD	d	12	4-30
32dd		SBD	d	12	4-30
33dd		LMD	d	12	4-31
34dd		STD	d	12	4-28
35dd		RAD	d	12	4-33
36dd		AOD	d	12	4-33
37dd		SOD	d	12	4-33
40dd		LDI	d	12	4-29
41dd		ADI	d	12	4-30
42dd		SBI	d	12	4-30
43dd		LMI	d	12	4-31
44dd		STI	d	12	4-28
45dd		RAI	d	12	4-33
46dd		AOI	d	12	4-33
47dd		SOI	d	12	4-33
50dd	mmmm	LDM	m,d	24	4-29
51dd	mmmm	ADM	m,d	24	4-30
52dd	mmmm	SBM	m,d	24	4-30
53dd	mmmm	LMM	m,d	24	4-32
54dd	mmmm	STM	m,d	24	4-29
55dd	mmmm	RAM	m,d	24	4-34
56dd	mmmm	AOM	m,d	24	4-34
57dd	mmmm	SOM	m,d	24	4-34
60dd		CRD	d	12	4-36
61dd	mmmm	†CRM	m,d	24	4-36
62dd		CWD	d	12	4-37

† A warning flag will be given if d is absent.

Octal Value		Machine Instruction		Length (bits)	Page
63dd	mmmm	†CWM	m,d	24	4-37
64dd	mmmm	†AJM	m,d	24	4-38
65dd	mmmm	†IJM	m,d	24	4-38
66dd	mmmm	†FJM	m,d	24	4-38
67dd	mmmm	†EJM	m,d	24	4-39
70dd		IAN	d	12	4-39
71dd	mmmm	†IAM	m,d	24	4-39
72dd		OAN	d	12	4-39
73dd	mmmm	†OAM	m,d	24	4-40
74dd		ACN	d	12	4-40
75dd		DCN	d	12	4-40
76dd		FAN	d	12	4-41
77dd	mmmm	†FNC	m,d	24	4-41

† A warning flag will be given if d is absent.

PSEUDO INSTRUCTIONS E

Instruction	Operation	Page No.
ABS	Declares absolute assembly	5-3
BASE	Declares mode of integers - octal or decimal	5-3
BSS	Allocates a block of storage	5-8
BSSZ	Allocates a zero-filled block of storage	5-8
DATA	Defines absolute data items	5-9
DIS	Defines display code data	5-9
DUP	Duplicates a sequence of code	5-22
EJECT	Ejects a page	5-21
END	Ends a subprogram	5-2
ENDD	Ends a DUP range	5-22
ENDIF	Ends a conditional range	5-18
ENDM	Ends a macro definition	6-7
ENTRY	Declares subprogram entry points	5-7
EQU	Equates a symbol to a value	5-8
ERR	Produces a fatal error flag	5-24
EXT	Defines symbols external to the subprogram	5-7
HERE	Calls for remote coding to be assembled	5-24
IDENT	Identifies beginning of a subprogram	5-2
IFxx	Compares two values for EQ, NE, GT, GE, LT, LE, and conditionally assembles a code sequence	5-13
IF	Tests a symbol for the attributes, absolute, relocatable, common, external, local SET, register, defined, and conditionally assembles a code sequence	5-14
IFPP,IFCP	Tests assembly environment (PP or CP)	5-14
IFC	Compares two character strings for equality	5-16
LCC	Loader Control	5-24
LIST	Declares assembly listing control parameters	5-19
LIT	Declares literals	5-10

Instruction	Operation	Page No.
LOC	Resets location counter	5-7
LOCAL	Declares symbols local to a macro	6-3
MACRO	Introduces a macro definition	6-1
macro name	calls a macro	6-1
MICRO	Defines a micro (character string)	7-2
OPDEF	Defines a macro	6-10
ORG	Resets origin counter	5-6
PERIPH	Declares a peripheral processor subprogram	5-3
REP	Declares loader-controlled code duplication	5-11
REPI	Declares loader-controlled code duplication	F-7
RMT	Introduces a sequence of remote code	5-23
SEGMENT	Produces CP and PP overlays at assembly time	5-4
SET	Equates a redefinable symbol to a value	5-9
SPACE	Spaces output listing	5-21
SST	Introduces system symbol definition	5-25
STOPDUP	Stops a DUP process	5-23
TITLE	Defines listing title or subtitle	5-21
USE	Names a block to receive subsequent code	5-4
VFD	Assigns data in variable byte sizes	5-10
XTEXT	Calls for text from an external source	5-25

RELOCATABLE SUBROUTINE FORMAT

The deck of one subprogram (subroutine) as it is output from an assembler or compiler comprises one logical record. Each logical record is made up of an indefinite number of tables. Each table is preceded by an identification word which specifies to the loader the procedure to be followed in loading the table. The identification word has the format:

CN		WC		LR	L
59	53	47	35	26	17 0

CN = Code number identifying type of data in table (text, entry points, external references, etc).

WC = Word count in table excluding identification word

LR = Method of relocation for the load address

L = Load address, 18-bits as defined for each type of table

LR and other relocation fields in the tables are nine bits long. Six of the nine are used currently; the other three are reserved for future expansion.

Prefix Table

The prefix table, if present, is the first table in a subroutine. It is bypassed by the loader. The prefix table is used by EDITLIB in constructing or modifying the SCOPE library. The format of the table is:

CN = 77_8 LR and L are ignored.

	name of subprogram	
word 1	59	17 0

The binary output from an assembly consists of all loader control cards (LCC) written as individual records, then an identification table of 14 words is written (77-table), followed by the deck. If errors occur in assembly, no binary output, except the 77-table and any LCC records, will appear.

For absolute programs, following the 77 table is another control word followed by the absolute program. This control word contains:

CP Programs: 5000 L_1L_2 ffff fftt tttt

L_1, L_2 = 00 for first overlay

= 01 for subsequent overlays

ffffff = origin -1 as specified on the IDENT line

tttttt = entry point address as specified on IDENT line

PP Programs: nnnn nn00 ffff 0000 cccc

nnnnnn = program name

ffff = origin -5 as specified on the IDENT line

cccc = program length (including this control word) in central memory: (program length+9)/5

Segment Overlays:

5000	L_1,	L_2	ffff ff	tt tttt
59	47	41	33	17 0

L_1, L_2 =0100 for all segment overlays

ffffff =origin -1 as specified on the segment line

tttttt =entry point address as specified on the segment line

L_1 =primary overlay level

L_2 =secondary overlay level

PIDL

Program identification and Length table contains the subprogram identification and declarations concerning common block allocation.

Identification Word

CN 34_8

LR Unused

L 0

word 1

name of subprogram	PL
59	17 0

PL Program length

words 2-WC

name of common block	BL
59	17 0

If blank common, name is 7 display code blank characters.

BL Block length

If WC=1, no common references appear in the program. Subprogram length is relevant only in the first PIDL table. All PIDL tables must appear before any other tables for a given subprogram. The names of common blocks may not be duplicated in a PIDL table. The list of common block names is called the Local Common Table (LCT). Since relocation of addresses relative to common blocks is designated by positions in LCT, the order of the common block names is significant.

The first word in the LCT is referred to as position 1.

ENTR

The entry point table contains a list of all the named entry points to the subprogram and its associated labeled common blocks. The ENTR table must immediately follow the PIDL table.

Identification Word

CN = 36_8

LR = Ignored

L = Ignored

Words 1 through WC

Each entry in the table is 2 words long. The first word contains the name of the entry point. The second word contains the location of the entry point.

first word

entry point name	
59	17 0

second word

	RL	LOC
59	26	17 0

RL = relocation of the address specified by LOC;

0 absolute, relative to RA (no relocation)

1 program relocatable

$3\text{-}77_8$ relative to common block M, where M is in position RL-2 of LCT. M must not refer to blank common.

LOC = address of entry point

TEXT

Text and data tables contain data comprising the subprogram and information necessary for properly relocating the data. The table consists of: an origin for the data, the data itself, and indicators describing relocation (if any) of the three possible locations in a data word which may refer to addresses in memory. TEXT tables may appear in any order and any numbers.

WC must be in the range 2 through 20_8.

Identification word

CN = 40_8

LR = relocation of load address (L)

0	absolute, relative to RA
1	relative to program origin
3-77_8	relative to labeled common block M; M is in position LR-2 of LCT. Values of 2 and n, where n refers to blank common, are not permitted.

L = load address. Initial location of data appearing in the second word of the table. L will be relocated using LR.

First Word

Relocation word consists of a series of 4-bit bytes describing the relocation of each of the three possible address references in a 60-bit data word. The first byte (bits 56-59) describes the relocation for the data word in the second word of the TEXT table, ctc. The number of relevant bytes and data words is determined by WC. Relocation is relative to program origin or the complement of the program origin (negative relocation). The value and relocation for each byte follows:

000X	no relocation
10XX	upper address, program relocation
11XX	upper address, negative relocation
010X	middle address, program relocation
011X	middle address, ncgative relocation
1X10	lower address, program relocation
1X11	lower address, negative relocation
0010	same as 1X10
0011	same as 1X11

The above designations permit independent and simultaneous relocation of both upper and lower addresses.

Words 2 through WC

Data words are loaded consecutively beginning at L. Their addresses are relocated as specified by the corresponding byte in the relocation word.

Note that with the text table all addresses are relocated absolute or relative to program origin, never relative to a labeled common block. As a result, addressing relative to labeled common for text must be accomplished through FILL tables.

FILL

The FILL table contains information necessary to relocate previously loaded address fields. References to common are relocated through this table. Program relocation may also be effected using the FILL table, although the usual method (with fewer words) is to use the TEXT table.

Identification Word

CN = 42_8

LR = 0

L = 0

Words 1 through WC

All remaining words are partitioned into sets of 30-bit contiguous bytes, each set is headed by one control byte and followed by an indefinite number of data bytes. The last byte may be zero. The control byte contains information concerning each of the subsequent data bytes until another control byte is encountered.

A zero byte is treated as a control byte in the format:

0		AR
29	8	0

AR is the relocation of the value in the address position of a word specified in the succeeding data bytes. AR has the value:

0	absolute, relative to RA (no relocation)
1	program relocation
2	negative relocation
$3\text{-}77_8$	relative to common block M where M is in position AR-2 of LCT.

One control byte suffices for several data bytes. The format of the data byte is:

1	P	RL	LOC
29	26	17	0

P = Position within word of address specified by RL and LOC.

10 upper

01 middle

00 lower

RL = Relocation of address specified by LOC.

RL has the same range of values as AR in the control byte except that 2 and any reference to blank common are illegal.

LOC = Address of data word to be modified.
The contents of address field position (P) at location LOC relative to RL is added to the origin as specified by AR in the control byte.

LINK

The LINK table indicates external references within the subprogram. Each reference to an external symbol must appear as an entry in LINK.

Identification Word

CN = 44_8

LR = Ignored

L = 0

All remaining words are partitioned into sets consisting of one 60-bit name word and a series of 30-bit contiguous data bytes indicating address positions which refer to the external symbol described in the name word. It is possible for the name word to be split between two computer words.

name of external symbol	
59	17 0

Names of external symbols (7 characters) must begin with a character for which the display code representation has a high order bit equal to zero. The data bytes have the form:

1	P	RL	LOC
29	26	17	0

P = Position within the word of the reference to the external symbol:

10	upper
01	middle
00	lower

R = Relocation of address specified by LOC

0	absolute, relative to RA
1	program relocation
3-77_8	relative to common block M where M is in position RL-2 of LCT.

LOC = Address of the word containing the reference to the external symbol

REPL — Replication Table

The REPL table permits the repetition of a block of data without requiring one word per location in a TEXT table.

Identification Word

CN = 43_8

LR = Ignored

L = 1 if replication is not to be deferred until all text is loaded. (Instant replication)

Words 1 through WC

Each entry in the table consists of two words in the format:

word 1	I		SR	S
word 2	C	B	DR	D
	59	41	26	17 ... 0

S	= Initial address of the source data, must be non-zero	
SR	= Relocation of the address specified by S.	
	0	Absolute, relative to RA
	1	Program relocation
	$3\text{-}77_8$	Relative to common block M, where M is in position SR-2 of LCT. M must not refer to blank common
D	= Initial address of destination of data	
DR	= Relocation of address specified by D; range of values same as SR-	
B	= Size of data block	
C	= Number of times data block is to be repeated	
I	= Increment to be added to D before each data block is repeated, first repetition of block is at D, second at D+I, etc. The data block (B-long) with origin at S is repeated C times beginning at D the first time, and beginning at the previous origin plus I thereafter.	
If C = 0	C is interpreted as 1	
If B = 0	B is interpreted as 1	
If I = 0	I is interpreted as B	
If D = 0	D is interpreted as S+B	

XFER — Transfer Table

The XFER table indicates the end of a subroutine and a pointer address.

Identification Word

CN = 46

LR = Ignored

L = Ignored

entry point name	
59	17 — 0

The entry point name need not be in the subprogram. If name is blank, there is no named XFER.

The location of the entry point is returned following a loader request. If a named XFER is encountered prior to an EXECUTE, control is transferred to that entry point. Otherwise, the job is aborted with the comment NO TRANSFER ADDRESS. If more than one subprogram has a named XFER, control is given to the last encountered XFER name.

SYSTEXT — Systems Text

Normally, systems text is derived from the library overlay named SYSTEXT, and is assembled prior to assembly of the source program, although this may be changed through the S option. Systems text overlays on the library look like loader overlays with the following control word:

5000 0101 0000 0000 0000

Data consists of coded lines. A minus zero word follows the last coded line.

Systems text can be deleted by using the S option with a dummy (non-existent) record name. A non-fatal loader message is produced when COMPASS attempts to load the overlay.

CENTRAL PROCESSOR INSTRUCTION EXECUTION TIMES G

Central Processor instruction execution times for the 6400, 6500 and 6600 systems are tabulated below. Instructions are arranged according to the functional units in which they are executed for the 6600 system. Time is counted from operand input to instruction result in the specified result register and includes readying the next instruction for execution. CM access time is not considered in increment instructions which result in memory references to read operands or store results. Instruction execution times are listed in minor cycles (one minor cycle = 100 nanoseconds); 4 minor cycles is an execution time of 400 nanoseconds.

INSTRUCTION EXECUTION TIMES: CENTRAL PROCESSOR

Octal Code	BRANCH UNIT	6400 6500	6600
00	STOP	-	-
01	RETURN JUMP to K	††††† 21	13
011	READ Extended Core Storage	††††	††††
012	WRITE Extended Core Storage	††††	††††
02	GO TO K + Bi†	13	14
030	GO TO K if Xj = zero	13	9†††
031	GO TO K if Xj ≠ zero	13	9†
032	GO TO K if Xj = positive	13	9†
033	GO TO K if Xj = negative	13	9†
034	GO TO K if Xj is in range ††	13	9†
035	GO TO K if Xj is out of range	††††† 13	9†
036	GO TO K if Xj is definite	13	9†
037	GO TO K if Xj is indefinite	13	9†
04	GO TO K if Bi = Bj†	13	8†
05	GO TO K if Bi = Bj†	13	8†
06	GO TO K if Bi Bj†	13	8†
07	GO TO K if Bi < Bj†	13	8†

†Tests made in Increment unit.

††Tests made in Long Add Unit.

†††Add 6 minor cycles to branch time for a branch to an instruction which is out of stack (no memory conflict considered); add 2 minor cycles for a no branch condition in the stack. Add 5 minor cycles for a no branch condition out of the stack.

††††Execution times for ECS operations depend on several factors.

†††††When jump condition is met include time to obtain new instruction word from storage and ready it for execution.

Octal Code	BOOLEAN UNIT	6400 6500	6600
10	TRANSMIT Xj to Xi	5	3
11	LOGICAL PRODUCT of Xj and Xk to Xi	5	3
12	LOGICAL SUM of Xj and Xk to Xi	5	3
13	LOGICAL DIFFERENCE of Xj and Xk to Xi	5	3
14	TRANSMIT Xk COMP. to Xi†	5	3
15	LOGICAL PRODUCT of Xj and Xk COMP. to Xi	5	3
16	LOGICAL SUM of Xj and Xk COMP. to Xi	5	3
17	LOGICAL DIFFERENCE of Xj and Xk COMP. to Xi	5	3
	SHIFT UNIT		
20	SHIFT Xi LEFT jk places	6	3
21	SHIFT Xi RIGHT jk places	6	3
22	SHIFT Xk NOMINALLY LEFT Bj places to Xi	6	3
23	SHIFT Xk NOMINALLY RIGHT Bj places to Xi	6	3
24	NORMALIZE Xk in Xi and Bj	7	4
25	ROUND AND NORMALIZE Xk in Xi and Bj	7	4
26	UNPACK Xk to Xi and Bj	7	3
27	PACK Xi from Xk and Bj	7	3
43	FORM jk MASK in Xi	6	3
	ADD UNIT		
30	FLOATING SUM of Xj and Xk to Xi	11	4
31	FLOATING DIFFERENCE of Xj and Xk to Xi	11	4
32	FLOATING DP SUM of Xj and Xk to Xi†	11	4
33	FLOATING DP DIFFERENCE of Xj and Xk to Xi	11	4
34	ROUND FLOATING SUM of Xj and Xk to Xi	11	4
35	ROUND FLOATING DIFFERENCE of Xj and Xk to Xi	11	4
	LONG ADD UNIT		
36	INTEGER SUM of Xj and Xk to Xi	6	3
37	INTEGER DIFFERENCE of Xj and Xk to Xi	6	3
	MULTIPLY UNIT††		
40	FLOATING PRODUCT of Xj and Xk to Xi	57	10
41	ROUND FLOATING PRODUCT of Xj and Xk to Xi	57	10
42	FLOATING DP PRODUCT of Xj and Xk to Xi	57	10

† COMP. = Complement; DP = Double Precision.

†† Duplexed units — instruction goes to free unit.

Octal Code	DIVIDE UNIT	6400 6500	6600
44	FLOATING DIVIDE Xj by Xk to Xi	56	29
45	ROUND FLOATING DIVIDE Xj by Xk to Xi	56	29
47	SUM of 1's in Xk to Xi	68	8
46	PASS	3	1
	INCREMENT UNIT†		
50	SUM of Aj and K to Ai	††	3
51	SUM of Bj and K to Ai	††	3
52	SUM of Xj and K to Ai	††	3
53	SUM of Xj and Bk to Ai	††	3
54	SUM of Aj and Bk to Ai	††	3
55	DIFFERENCE of Aj and Bk to Ai	††	3
56	SUM of Bj and Bk to Ai	††	3
57	DIFFERENCE of Bj and Bk to Ai	††	3
60	SUM of Aj and K to Bi	5	3
61	SUM of Bj and K to Bi	5	3
62	SUM of Xj and K to Bi	5	3
63	SUM of Xj and Bk to Bi	5	3
64	SUM of Aj and Bk to Bi	5	3
65	DIFFERENCE of Aj and Bk to Bi	5	3
66	SUM of Bj and Bk to Bi	5	3
67	DIFFERENCE of Bj and Bk to Bi	5	3
70	SUM of Aj and K to Xi	6	3
71	SUM of Bj and K to Xi	6	3
72	SUM of Xj and K to Xi	6	3
73	SUM of Xj and Bk to Xi	6	3
74	SUM of Aj and Bk to Xi	6	3
75	DIFFERENCE of Aj and Bk to Xi	6	3
76	SUM of Bj and Bk to Xi	6	3
77	DIFFERENCE of Bj and Bk to Xi	6	3

†Duplexed units - instruction goes to free unit.

††i = 0 execution time is 6 minor cycles;
i = 1-5 time is 12 minor cycles;
i = 6 or 7 time is 10 minor cycles.

PERIPHERAL AND CONTROL PROCESSOR

The execution time of PP and CP instructions is influenced by the following factors:

> Number of memory references — indirect addressing and indexed addressing require an extra memory reference. Instructions in 24-bit format require an extra reference to read m.
>
> Number of words to be transferred — in I/O instructions and in references to CM the execution times vary with the number of words to be transferred. The maximum theoretical rate of flow is one word/major cycle. I/O word rates depend upon the speed of external equipments, normally much slower than the computer.
>
> References to CM may be delayed if there is conflict with CP memory requests.
>
> Following an exchange jump instruction, no memory references (nor other exchange jump instructions) may be made until the CP has completed thc exchange jump.

PERIPHERAL AND CONTROL PROCESSOR INSTRUCTION EXECUTION TIMES
(6400, 6500 and 6600)

Octal Code	Name	Time† (Major Cycles)
00	Pass	1
01	Long jump to m + (d)	2-3
02	Return jump to m + (d)	3-4
03	Unconditional jump d	1
04	Zero jump d	1
05	Nonzero jump d	1
06	Plus jump d	1
07	Minus jump d	1
10	Shift d	1
11	Logical difference d	1
12	Logical product d	1
13	Selective clear d	1
14	Load d	1
15	Load complement d	1
16	Add d	1
17	Subtract d	1

†A major cycle is 1000 nanoseconds.

Octal Code	Name	Time† (Major Cycles)
20	Load dm	2
21	Add dm	2
22	Logical product dm	2
23	Logical difference dm	2
24	Pass	1
25	Pass	1
26	Exchange jump	min. 2
27	Read program address	1
30	Load (d)	2
31	Add (d)	2
32	Subtract (d)	2
33	Logical difference (d)	2
34	Store (d)	2
35	Replace add (d)	3
36	Replace add one (d)	3
37	Replace subtract one (d)	3
40	Load ((d))	3
41	Add ((d))	3
42	Subtract ((d))	3
43	Logical difference ((d))	3
44	Store ((d))	3
45	Replace add ((d))	4
46	Replace add one ((d))	4
47	Replace subtract one ((d))	4
50	Load (m + (d))	3-4
51	Add (m + (d))	3-4
52	Subtract (m+ (d))	3-4
53	Logical difference (m + (d))	3-4
54	Store (m + (d))	3-4
55	Replace add (m + (d))	4-5
56	Replace add one (m + (d))	4-5
57	Replace subtract one (m + (d))	4-5
60	Central read from (A) to d	min. 6
61	Central read (d) words from (A) to m	5 plus 5/word

†A major cycle is 1000 nanoseconds.

CONVERSION TABLES

OCTAL/DECIMAL INTEGER CONVERSION TABLE

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0089	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0138	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0180	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0218	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0286	0287	0268	0289	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0480	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0358	0357	0358	0359
0550	0360	0361	0362	0383	0364	0365	0366	0367
0580	0368	0369	0370	0371	0372	0373	0374	0375
0570	0378	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0480	0401	0402	0403	0404	0405	0406	0407
0830	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0850	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0482	0483
0720	0484	0465	0486	0487	0488	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0488	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

0000	0000
to	to
0777	0511
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24578
70000 -	28672

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0588	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0811	0812	0613	0614	0615
1150	0818	0617	0618	0619	0620	0621	0622	0623
1160	0824	0825	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0838	0639
1200	0840	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0852	0653	0654	0855
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0865	0666	0667	0668	0689	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0882	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0893	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0748	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0758	0757	0758	0759
1370	0760	0781	0782	0783	0764	0765	0788	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0798	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0808	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0838	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0908	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0827
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0848	0949	0850	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0983	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0978	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

1000	0512
to	to
1777	1023
(Octal)	(Decimal)

OCTAL/DECIMAL INTEGER CONVERSION TABLE (Cont'd)

2000 to 2777 (Octal) — 1024 to 1535 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1048	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2100	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1180	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1284	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1383	1364	1385	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2800	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1519	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

3000 to 3777 (Octal) — 1536 to 2047 (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1580	1561	1562	1583	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1840	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1893	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1787
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1989	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

OCTAL/DECIMAL INTEGER CONVERSION TABLE (Cont'd)

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2178	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2281	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2298	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2328	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2388	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

4000 to 4777 (Octal)   2048 to 2559 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2588	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2863
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2891	2892	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2780	2761	2762	2763	2764	2765	2766	2767
5320	2768	2789	2770	2771	2772	2773	2774	2775
5330	2778	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2888	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

5000 to 5777 (Octal)   2560 to 3071 (Decimal)

OCTAL/DECIMAL INTEGER CONVERSION TABLE (Cont'd)

6000 to 6777 (Octal) — 3072 to 3583 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3182	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
8250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
8400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3382	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
8860	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
8770	3576	3577	3578	3579	3580	3581	3582	3583

7000 to 7777 (Octal) — 3584 to 4095 (Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3496	3497	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3786	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3800	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

OCTAL/DECIMAL FRACTION CONVERSION TABLE

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

OCTAL/DECIMAL FRACTION CONVERSION TABLE (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	000203	.000499	.000303	.000743
000004	.000015	.000104	.000259	000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000008	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000081	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000318	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000028	.000083	.000126	000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	000235	000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	000679	.000362	.000923
000063	.000194	.000183	.000438	.000263	.000882	.000363	.000926
000064	.000198	.000164	.000442	.000264	.000686	000364	.000930
000065	.000202	.000165	.000446	.000285	.000690	000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	000457	.000270	.000701	.000370	.000946
000071	.000217	.000171	.000461	.000271	000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000278	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

OCTAL/DECIMAL FRACTION CONVERSION TABLE (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 853 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 228 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 358 839 400 250 464 877 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 587 826 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 982 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 908 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625

DECIMAL/BINARY POSITION TABLE

Largest Decimal Integer	Decimal Digits Req'd*	Number of Binary Digits	Largest Decimal Fraction
1		1	.5
3		2	75
7		3	.875
15	1	4	.937 5
31		5	.968 75
63		6	.984 375
127	2	7	.992 187 5
255		8	.996 093 75
511		9	.998 046 875
1 023	3	10	.999 023 437 5
2 047		11	999 511 718 75
4 095		12	.999 755 859 375
8 191		13	.999 877 929 687 5
16 383	4	14	.999 938 964 843 75
32 767		15	.999 969 482 421 875
65 535		16	.999 984 741 210 937 5
131 071	5	17	.999 992 370 605 468 75
282 143		18	.999 996 185 302 734 375
524 287		19	.999 998 092 651 387 187 5
1 048 575	6	20	.999 999 046 325 683 593 75
2 097 151		21	999 999 523 182 841 796 875
4 194 303		22	.999 999 761 581 420 898 437 5
8 388 807		23	.999 999 880 790 710 449 218 75
18 777 215	7	24	.999 999 940 395 355 244 809 375
33 554 431		25	.999 999 970 197 677 612 304 687 5
67 108 863		26	.999 999 985 098 838 806 152 343 75
134 217 727	8	27	.999 999 992 549 419 403 078 171 875
268 435 455		28	.999 999 996 274 709 701 538 085 937 5
538 870 911		29	.999 999 998 137 354 850 769 042 968 75
1 073 741 823	9	30	.999 999 999 068 677 425 384 521 484 375
2 147 483 647		31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 787 169 356 348 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 878 173 065 185 546 875
17 179 869 183	10	34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 367		35	.999 999 999 970 896 169 543 266 296 386 718 75
88 719 476 735		36	.999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471	11	37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943		38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 169 921 875
1 099 511 627 775	12	40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103		42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415	13	44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 863		46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327	14	47	.999 999 999 999 992 894 572 642 398 998 141 288 757 324 218 75
281 474 976 710 655		48	.999 999 999 999 996 447 286 321 199 499 070 644 378 662 109 375
562 949 953 421 311		49	.999 999 999 999 998 223 643 160 599 749 535 322 189 331 054 687 5
1 125 899 906 842 623	15	50	.999 999 999 999 999 111 821 580 299 874 767 661 094 665 527 343 75
2 251 799 813 685 247		51	.999 999 999 999 999 555 910 790 149 937 383 830 547 332 763 671 875
4 503 599 627 370 495		52	.999 999 999 999 999 777 955 395 074 968 691 915 273 666 381 835 937 5
9 007 199 254 740 991		53	.999 999 999 999 999 888 977 697 537 484 345 957 636 833 190 917 968 75
18 014 398 509 481 983	16	54	.999 999 999 999 999 944 488 848 768 742 172 978 818 416 595 458 984 375
36 028 797 018 963 967		55	.999 999 999 999 999 972 244 424 384 371 086 489 409 208 297 729 492 187 5
72 057 594 037 927 935		56	.999 999 999 999 999 986 122 212 192 185 543 244 704 604 148 864 746 093 75
144 115 188 075 855 871	17	57	.999 999 999 999 999 993 061 106 096 092 771 622 352 302 074 432 373 046 875
288 230 376 151 711 743		58	.999 999 999 999 999 998 530 553 048 046 385 811 176 151 037 216 186 523 437 5
576 460 752 303 423 487		59	.999 999 999 999 999 998 265 276 524 023 192 905 588 075 518 608 093 261 718 75
1 152 921 504 606 846 975	18	60	.999 999 999 999 999 999 132 638 262 011 596 452 794 037 759 304 046 630 859 375

*Larger numbers within a digit group should be checked for exact number of decimal digits required.

Examples of use:

1. *Q.* What is the largest decimal value that can be expressed by 36 binary digits?
   A. 68,719,476,735.
2. *Q.* How many decimal digits will be required to express a 22-bit number?
   A. 7 decimal digits.

CONSTANTS

π	= 3.14159 26535 89793 23846 26433 83279 50
$\sqrt{3}$	= 1.732 050 807 569
$\sqrt{10}$	= 3.162 277 660 1683
e	= 2.71828 18284 59045 23536
ln 2	= 0.69314 71805 599453
ln 10	= 2.30258 50929 94045 68402
$\log_{10} 2$	= 0.30102 99956 63981
$\log_{10} e$	= 0.43429 44819 03251 82765
$\log_{10} \log_{10} e$	= 9.63778 43113 00537 − 10
$\log_{10} \pi$	= 0.49714 98726 94133 85435
1 degree	= 0.01745 32925 19943 radians
1 radian	= 57.29577 95131 degrees
$\log_{10}(5)$	= 0.69897 00043 36019
7!	= 5040
8!	= 40320
9!	= 362,880
10!	= 3,628,800
11!	= 39,916,800
12!	= 479,001,600
13!	= 6,227,020,800
14!	= 87,178,291,200
15!	= 1,307,674,368,000
16!	= 20,922,789,888,000
$\frac{\pi}{180}$	= 0.01745 32925 19943 29576 92369 07684 9
$\left(\frac{\pi}{2}\right)^2$	= 2.4674 01100 27233 96
$\left(\frac{\pi}{2}\right)^3$	= 3.8757 84585 03747 74
$\left(\frac{\pi}{2}\right)^4$	= 6.0880 68189 62515 20
$\left(\frac{\pi}{2}\right)^5$	= 9.5631 15149 54004 49
$\left(\frac{\pi}{2}\right)^6$	= 15.0217 06149 61413 07
$\left(\frac{\pi}{2}\right)^7$	= 23.5960 40842 00618 62
$\left(\frac{\pi}{2}\right)^8$	= 37.0645 72481 52567 57
$\left(\frac{\pi}{2}\right)^9$	= 58.2208 97135 63712 59
$\left(\frac{\pi}{2}\right)^{10}$	= 91.4531 71363 36231 53
$\left(\frac{\pi}{2}\right)^{11}$	= 143.6543 05651 31374 95
$\left(\frac{\pi}{2}\right)^{12}$	= 225.6516 55645 350
$\left(\frac{\pi}{2}\right)^{13}$	= 354.4527 91822 91051 47
$\left(\frac{\pi}{2}\right)^{14}$	= 556.7731 43417 624

CONSTANTS (Cont'd)

π^2	=	9.86960 44010 89358 61883 43909 9988
$2\pi^2$	=	19.73920 88021 78717 23766 87819 9976
$3\pi^2$	=	29.60881 32032 68075 85680 31729 9964
$4\pi^2$	=	39.47841 76043 57434 47533 75639 9952
$5\pi^2$	=	49.34802 20054 46793 09417 19549 9940
$6\pi^2$	=	59.21762 64065 36151 71300 63459 9928
$7\pi^2$	=	69.08723 08076 25510 33184 07369 9916
$8\pi^2$	=	78.95683 52087 14868 95067 51279 9904
$9\pi^2$	=	88.82643 96098 04227 56950 95189 9892

$\sqrt{2}$	=	1.414 213 562 373 095 048 801 688
$1 + \sqrt{2}$	=	2.414 213 562 373 095 048 801 688
$(1 + \sqrt{2})^2$	=	5.828 427 124 746 18
$(1 + \sqrt{2})^4$	=	33.970 562 748 477 08
$(1 + \sqrt{2})^6$	=	197.994 949 366 116 30
$(1 + \sqrt{2})^8$	=	1153.999 133 448 220 72
$(1 + \sqrt{2})^{10}$	=	6725.999 851 323 208 02
$(1 + \sqrt{2})^{12}$	=	39201.999 974 491 027 40
$(1 + \sqrt{2})^{14}$	=	228485.999 995 622 956 38
$(1 + \sqrt{2})^{16}$	=	1331713.999 999 246 711
$(1 + \sqrt{2})^{18}$	=	7761797.999 999 884 751

Sin .5	=	0.47942 55386 04203
Cos .5	=	0.87758 25618 90373
Tan .5	=	0.54630 24898 43790
Sin 1	=	0.84147 09848 07896
Cos 1	=	0.54030 23058 68140
Tan 1	=	1.55740 77246 5490
Sin 1.5	=	0.99749 49866 04054
Cos 1.5	=	0.07073 72016 67708
Tan 1.5	=	14.10141 99471 707

DIVIDE

$(\pm \infty) \div (\pm \infty)$ = 177700. . . 00

$(\ \infty) \div (\ N\ )$ = 377700. . . 00

$(\ \infty) \div (-N\ )$ = 400000. . . 00

$(-\infty) \div (\ N\ )$ = 400000. . . 00

$(\pm 0\ ) \div (\pm \infty)$ = 000000. . . 00

$(\pm 0\ ) \div (\pm N)$ = 000000. . . 00

$(\pm N) \div (\pm \infty)$ = 000000. . . 00

$(\ N) \div (\ 0\ )$ = 377700. . . 00

$(-N) \div (\ 0\ )$ = 400000. . . 00

$(\ N) \div (-0\ )$ = 400000. . . 00

$(-N) \div (-0\ )$ = 377700. . . 00

$(\pm$ Indef.$) \div (\pm N)$ = 177700. . . 00

$(\pm$ Indef.$) \div (\pm \infty)$ = 177700. . . 00

$(\pm$ Indef.$) \div (\pm 0\ )$ = 177700. . . 00

Underflow:	#	= 000000. . . 00
Overflow:	(right shift & sign record)	= 4000 (coefficient = coefficient X_j coefficient X_k)
	(right shift & sign record)	= 3777 (coefficient = coefficient X_j coefficient X_k)

\# Right shift one does not take the exponent out of underflow

NORMALIZE

$(+\infty)$ = 3777XX. . . XX $\quad B_j$ = 000000

$(-\infty)$ = 4000XX. . . XX $\quad B_j$ = 000000

$(\pm$ Indef.$)$ = 1777XX. . . XX $\quad B_j$ = 000000

Underflow = 0000. . . 00 $\quad B_j$ = Shift count

INDEFINITE FORMS

FLOATING ADD

$(+\infty) + (+\infty) = 377700\ldots00$

$(+\infty) + (-\infty) = 177700\ldots00$

$(-\infty) + (-\infty) = 400000\ldots00$

$(-\infty) + (+\infty) = 177700\ldots00$

$(+\infty) - (+\infty) = 177700\ldots00$

$(+\infty) - (-\infty) = 377700\ldots00$

$(-\infty) - (+\infty) = 400000\ldots00$

$(-\infty) - (-\infty) = 177700\ldots00$

$(+\infty) \pm (\pm N) = 377700\ldots00$

$(-\infty) \pm (\pm N) = 400000\ldots00$

$(\pm \text{Indef.}) \pm (\pm N) = 177700\ldots00$

$(\pm \text{Indef.}) \pm (\pm \infty) = 177700\ldots00$

$(\pm \text{Indef.}) \pm (\pm 0) = 177700\ldots00$

Underflow = 0000 (coefficient = coefficient $X_j \pm$ coefficient X_k)

Overflow on right shift one = 3777XXX...XX (coefficient positive)
4000XXX...XX (coefficient negative)

MULTIPLY

$(+\infty) \cdot (+\infty) = 377700\ldots00$

$(+\infty) \cdot (-\infty) = 400000\ldots00$

$(\pm\infty) \cdot (\pm 0) = 177700\ldots00$

$(\pm 0) \cdot (\pm 0) = 000000\ldots00$

$(\pm 0) \cdot (\pm N) = 000000\ldots00$

$(\pm \text{Indef.}) \cdot (\pm N) = 177700\ldots00$

$(\pm \text{Indef.}) \cdot (\pm \infty) = 177700\ldots00$

$(\pm \text{Indef.}) \cdot (\pm 0) = 177700\ldots00$

Underflow: (no left shift one) = 000000...00

(left shift one & sign record) = 7777 (coefficient = coefficient X_j coefficient X_k)

(left shift one & no sign record) = 0000 (coefficient = coefficient X_j coefficient X_k)

Overflow: # (sign record) = 40000...00

(no sign record) = 37700...00

Left shift one does not take the exponent out of overflow

SUPPLEMENT TO TABLE OF INDEFINITE FORMS
(Coefficient Fields for Indefinite Operands in X_j and/or X_k May Be Any Value in Any Flt. Pt. Unit)

FLOATING ADD UNIT USING 30, 31, 34 or 35 INSTRUCTION

X_j		X_k		X_i
37770000000000000000	+	37770000000000000000	=	37770000000000000000
37770000000000000000	+	40000000000000000000	=	17770000000000000000
40000000000000000000	+	40000000000000000000	=	40000000000000000000
40000000000000000000	+	37770000000000000000	=	17770000000000000000
37770000000000000000	-	37770000000000000000	=	17770000000000000000
37770000000000000000	-	40000000000000000000	=	37770000000000000000
40000000000000000000	-	37770000000000000000	=	40000000000000000000
40000000000000000000	-	40000000000000000000	=	17770000000000000000
37770000000000000000	+	17206000000000000000	=	37770000000000000000
37770000000000000000	+	60571777777777777777	=	37770000000000000000
37770000000000000000	-	17206000000000000000	=	37770000000000000000
37770000000000000000	-	60571777777777777777	=	37770000000000000000
40000000000000000000	+	17257000000000000000	=	40000000000000000000
40000000000000000000	+	60520777777777777777	=	40000000000000000000
40000000000000000000	-	17257000000000000000	=	40000000000000000000
40000000000000000000	-	60520777777777777777	=	40000000000000000000
17770000000000000000	+	16204500000000000000	=	17770000000000000000
17770000000000000000	+	61573277777777777777	=	17770000000000000000
60000000000000000000	+	16204500000000000000	=	17770000000000000000
60000000000000000000	+	61573277777777777777	=	17770000000000000000
17770000000000000000	-	16204500000000000000	=	17770000000000000000
17770000000000000000	-	61573277777777777777	=	17770000000000000000
60000000000000000000	-	16204500000000000000	=	17770000000000000000
60000000000000000000	-	61573277777777777777	=	17770000000000000000
17770000000000000000	+	37770000000000000000	=	17770000000000000000
17770000000000000000	+	40000000000000000000	=	17770000000000000000
60000000000000000000	+	37770000000000000000	=	17770000000000000000
60000000000000000000	+	40000000000000000000	=	17770000000000000000
17770000000000000000	-	37770000000000000000	=	17770000000000000000
17770000000000000000	-	40000000000000000000	=	17770000000000000000

FLOATING ADD (Cont'd)

X_j		X_k		X_i
60000000000000000000	-	37770000000000000000	=	17770000000000000000
60000000000000000000	-	40000000000000000000	=	17770000000000000000
37765400000000000000	+	37764000000000000000	=	37774600000000000000
40012377777777777777	+	40013777777777777777	=	40003177777777777777

FLOATING ADD UNIT USING 32 or 33 INSTRUCTION

00574320000000000000	+	00575400000000000000	=	00004750000000000000
77203457777777777777	+	77202377777777777777	=	77773027777777777777
00564320000000000000	+	00555400000000000000	=	00000000000000000000
77213457777777777777	+	77222377777777777777	=	00000000000000000000

MULTIPLY UNIT USING 40 or 41 INSTRUCTION

X_j		X_k		X_i
37770000000000000000	·	57773177777777777777	=	40000000000000000000
37770000000000000000	·	20004600000000000000	=	37770000000000000000
40000000000000000000	·	20004600000000000000	=	40000000000000000000
40000000000000000000	·	57773177777777777777	=	37770000000000000000
37770000000000000000	·	37770000000000000000	=	37770000000000000000
37770000000000000000	·	40000000000000000000	=	40000000000000000000
37770000000000000000	·	00000000000000000000	=	17770000000000000000
37770000000000000000	·	77777777777777777777	=	17770000000000000000
40000000000000000000	·	00000000000000000000	=	17770000000000000000
40000000000000000000	·	77777777777777777777	=	17770000000000000000
00000000000000000000	·	17154370000000000000	=	00000000000000000000
77777777777777777777	·	17154370000000000000	=	00000000000000000000
00000000000000000000	·	60623407777777777777	=	00000000000000000000
77777777777777777777	·	60623407777777777777	=	00000000000000000000
17770000000000000000	·	20606543000000000000	=	17770000000000000000
17770000000000000000	·	57171234777777777777	=	17770000000000000000
60000000000000000000	·	20606543000000000000	=	17770000000000000000
60000000000000000000	·	57171234777777777777	=	17770000000000000000
17770000000000000000	·	37770000000000000000	=	17770000000000000000
17770000000000000000	·	40000000000000000000	=	17770000000000000000
60000000000000000000	·	37770000000000000000	=	17770000000000000000
60000000000000000000	·	40000000000000000000	=	17770000000000000000
00305000000000000000	·	16277000000000000000	=	00000000000000000000
00305000000000000000	·	61500777777777777777	=	00000000000000000000
77472777777777777777	·	16277000000000000000	=	00000000000000000000
77472777777777777777	·	61500777777777777777	=	00000000000000000000
07214000000000000000	·	07777000000000000000	=	00007000000000000000
70563777777777777777	·	07777000000000000000	=	77770777777777777777
30007000000000000000	·	27174000000000000000	=	37770000000000000000
30007000000000000000	·	50603777777777777777	=	40000000000000000000

DIVIDE UNIT USING 44 OR 45 INSTRUCTION

X_j		X_k		X_i
00000000000000000000	/	00000000000000000000	=	17770000000000000000
00000000000000000000	/	77777777777777777777	=	17770000000000000000
77777777777777777777	/	00000000000000000000	=	17770000000000000000
77777777777777777777	/	77777777777777777777	=	17770000000000000000
37770000000000000000	/	37770000000000000000	=	17770000000000000000
37770000000000000000	/	40000000000000000000	=	17770000000000000000
40000000000000000000	/	37770000000000000000	=	17770000000000000000
40000000000000000000	/	40000000000000000000	=	17770000000000000000
37770000000000000000	/	20424321000000000000	=	37770000000000000000
37770000000000000000	/	57353456777777777777	=	40000000000000000000
40000000000000000000	/	20424321000000000000	=	40000000000000000000
40000000000000000000	/	57353456777777777777	=	37770000000000000000
00000000000000000000	/	37770000000000000000	=	00000000000000000000
00000000000000000000	/	40000000000000000000	=	00000000000000000000
77777777777777777777	/	37770000000000000000	=	00000000000000000000
77777777777777777777	/	40000000000000000000	=	00000000000000000000
00000000000000000000	/	17347560000000000000	=	00000000000000000000
00000000000000000000	/	60430217777777777777	=	00000000000000000000
77777777777777777777	/	17347560000000000000	=	00000000000000000000
77777777777777777777	/	60430217777777777777	=	00000000000000000000
16717400000000000000	/	37770000000000000000	=	00000000000000000000
16717400000000000000	/	40000000000000000000	=	00000000000000000000
61060377777777777777	/	37770000000000000000	=	00000000000000000000
61060377777777777777	/	40000000000000000000	=	00000000000000000000
32044540000000000000	/	00000000000000000000	=	37770000000000000000
45733237777777777777	/	00000000000000000000	=	40000000000000000000
20615567000000000000	/	77777777777777777777	=	40000000000000000000
57162210777777777777	/	77777777777777777777	=	37770000000000000000
17770000000000000000	/	17367540000000000000	=	17770000000000000000
17770000000000000000	/	60410237000000000000	=	17770000000000000000
60000000000000000000	/	17756677000000000000	=	17770000000000000000
60000000000000000000	/	60021100777777777777	=	17770000000000000000
17770000000000000000	/	37770000000000000000	=	17770000000000000000

DIVIDE (Cont'd)

X_j		X_k		X_i
17770000000000000000	/	40000000000000000000	=	17770000000000000000
60000000000000000000	/	37770000000000000000	=	17770000000000000000
60000000000000000000	/	40000000000000000000	=	17770000000000000000
07776000000000000000	/	27204000000000000000	=	00000000000000000000
30006000000000000000	/	07214000000000000000	=	37776000000000000000
47771777777777777777	/	07214000000000000000	=	40001777777777777777

NORMALIZE

X_k	B_j	X_i
37770043200000000000	000000	37770043200000000000
40007734577777777777	000000	40007734577777777777
17770002100000000000	000000	17770002100000000000
60007775677777777777	000000	60007775677777777777
00000000000000000000	000060	00000000000000000000
* 00000000000000000000	000060	00000000000000000000
00040006000000000000	000011	00000000000000000000
77777777777777777777	000060	00000000000000000000
* 77777777777777777777	000060	00000000000000000000
77737777777777777777	000011	00000000000000000000
20000000000000000000	000060	00000000000000000000
* 20000000000000000000	000060	17174000000000000000
57777777777777777777	000060	00000000000000000000
* 57777777777777777777	000060	60603777777777777777

* Results due to rounded normalize

INDEX

COMMENT SHEET

CONTROL DATA
CORPORATION

TITLE: 6400/6500/6600 COMPASS Reference Manual

PUBLICATION NO. 60190900 REVISION B

Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

FROM NAME: ______________________ POSITION: ______________________

BUSINESS
ADDRESS: ____

____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

CUT ON THIS LINE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA 94304

CUT ON THIS LINE

FOLD FOLD

STAPLE STAPLE

CONTROL DATA

1/2 3/4 1 1-1/4

►► CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60190900

Litho in U.S.A.